EXTENDED MODELS OF FINITE AUTOMATA

by

Özlem Salehi Köken

B.S., Mathematics, Boğaziçi University, 2011

M.S., Computer Engineering, Boğaziçi University, 2013

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2019

To the memory of my grandfather Hasan Salehi,
my great-grandmother Gülizar Burhanetin, and
my grandmother Mader Burhanettin...

ACKNOWLEDGEMENTS

First of all I would like to express my gratitude to my supervisor Prof. A. C. Cem Say for his guidance and support throughout my journey at Boğaziçi University. He has always been a source of inspiration for me and it is a great pleasure to be his student. I would also like to thank Abuzer Yakaryılmaz for his significant contributions to my thesis. His enthusiasm has always motivated and encouraged me.

I would like to thank my thesis committee members Prof. Can Özturan and Assoc. Prof. Özlem Beyarslan for their valuable feedbacks over the years and devoting their time to this research. I would like to thank Prof. Sema Fatma Oktuğ and Assoc. Prof. Tolga Ovatman for kindly accepting to be in my thesis jury and for their helpful comments.

I am grateful to Dr. Flavio D'Alessandro for our joint works and for sharing his knowledge. It was a great experience to work with him. I would like to thank professors Ryan O'Donnell, Alexandre Borovik and Igor Potapov for their helpful answers to my questions.

My gratitudes go to all members of the Department of Computer Engineering for providing such a nice environment. I would like to especially thank Prof. Pınar Yolum Birbil and Prof. Cem Ersoy for their kindness and support. I want to thank all past and current members of BM26 with whom I had the chance of working with.

I would like to thank Pelin Gürel for her valuable friendship all through these years.

I would like to offer my deepest gratitude to my grandfather İbrahim Burhanettin, my aunt Perihan Burhanettin, and my parent-in-law Nebahat Köken for their endless love and faith in me. I want to thank my sister Meltem Salehi for always being cheerful and motivating me. I am indebted to my parents Feryal and Mecid Salehi for all their

efforts and generosity which made this thesis possible. Finally I would like to express my love and gratitude to my husband Oktay Köken for his patience, guidance and encouragement and to my lovely daughter Eda for bringing joy to my life. Without them, it would have no meaning.

This work is supported by Boğaziçi University Research Fund Grant Number 11760 and by Scientific and Technical Research Council of Turkey (TÜBİTAK) BİDEB Scholarhip program.

ABSTRACT

EXTENDED MODELS OF FINITE AUTOMATA

Many of the numerous automaton models proposed in the literature can be regarded as a finite automaton equipped with an additional storage mechanism. In this thesis, we focus on two such models, namely the finite automata over groups and the homing vector automata.

A finite automaton over a group G is a nondeterministic finite automaton equipped with a register that holds an element of the group G. The register is initialized to the identity element of the group and a computation is successful if the register is equal to the identity element at the end of the computation after being multiplied with a group element at every step. We investigate the language recognition power of finite automata over integer and rational matrix groups and reveal new relationships between the language classes corresponding to these models. We examine the effect of various parameters on the language recognition power. We establish a link between the decision problems of matrix semigroups and the corresponding automata. We present some new results about valence pushdown automata and context-free valence grammars.

We also propose the new homing vector automaton model, which is a finite automaton equipped with a vector that can be multiplied with a matrix at each step. The vector can be checked for equivalence to the initial vector and the acceptance criterion is ending up in an accept state with the value of the vector being equal to the initial vector. We examine the effect of various restrictions on the model by confining the matrices to a particular set and allowing the equivalence test only at the end of the computation. We define the different variants of the model and compare their language recognition power with that of the classical models.

ÖZET

GÜÇLENDİRİLMİŞ SONLU DURUMLU MAKİNE MODELLERİ

Literatürde ortaya sürülmüş olan pek çok makine, bir sonlu durumlu makinenin ek bir hafıza ünitesi ile güçlendirilmiş hali olarak düşünülebilir. Bu tezde, bu makinelerden ikisine, gruplar üzerinde tanımlı sonlu durumlu makinelere ve eve dönen vektör makinelerine odaklanılmıştır.

G grubu üzerinde tanımlı bir makine, ek hafıza ünitesinde G grubundan bir elemanı tutma hakkına sahip, belirlenimci olmayan bir sonlu durumlu makinedir. Başlangıçta hafıza ünitesinin değeri G grubunun birim elemanıdır. Bir hesaplamanın başarılı sayılabilmesi için hafıza ünitesinin değeri, her adımda grubun bir elemanıyla çarpıldıktan sonra bitimde grubun birim elemanına eşit olmalıdır. Bu çalışmada tam sayılı ve rasyonel sayılı matris grupları üzerinde tanımlanan sonlu durumlu makinelerin tanıdıkları dil sınıfları incelenmiştir. Çeşitli parametrelerin makinelerin tanıma gücünü nasıl etkilediği araştırılmıştır. Matris yarıgruplarının karar verme problemleri ile ilintili makinelerinki arasında bir bağ kurulmuştur. Grup üzerinde tanımlı makinelerle ilişkili olan bazı modellerle ilgili yeni sonuçlar elde edilmiştir.

Yeni tanımladığımız eve dönen vektör makinesi, bir sonlu durumlu makinenin bir vektörle güçlendirilmesi ve bu vektöre her adımda bir matrisle çarpılma hakkı verilmesiyle ortaya çıkmıştır. Vektörün başlangıç vektörüne eşit olup olmadığı kontrol edilebilir ve makinenin kabul şartı, hesaplama bittiğinde vektörün başlangıç vektörüne eşit olması ve kabul durumlarından birinde bulunulmasıdır. Kullanılan matris kümesi sınırlanarak ve vektörün eşitlik kontrolünün sadece sonda gerçekleşmesine izin verilerek, farklı kısıtlamaların makineye olan etkisi incelenmiştir. Makinenin çeşitli sürümlerinin dil tanıma gücüyle klasik modellerin dil tanıma gücü karşılaştırılmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

1	The identity element of a group/monoid
$\|A\|$	The cardinality of a set A
$\mathcal{A}$	A pushdown automaton
$A[i,j]$	The entry in the i'th row and the j'th column of the matrix A
$A_L(n)$	The maximum k such that there exist k distinct strings that are pairwise n-dissimilar for L
B	The bicyclic monoid
$B_G^A(n)$	The set of all elements in G which can be represented by a word of length at most n
$BS(m,n)$	The Baumslag Solitar group
$\mathcal{C}$	A counter automaton
$\mathcal{D}$	The set of head directions
e	The identity element of a group/monoid
$e_m(w)$	The base-10 number encoded by w in base-m
$\mathcal{E}$	An extended finite automaton
$\mathcal{F}$	A finite automaton
$\mathbf{F}_r$	The free group of rank r
G	A group
$\mathcal{G}$	A context-free grammar
$\mathfrak{G}$	A context-free valence grammar
$g_G(n)$	The growth function of a group G
$GL(n,\mathbb{Q})$	The general linear group of order n over the field of rationals
$GL(n,\mathbb{Z})$	The general linear group of order n over the field of integers
$\mathbf{H}$	The discrete Heisenberg group
I	Identity matrix
$\mathtt{I}$	An ideal
L	A language
$\bar{L}$	The complement of the language L

$L(.)$	The language generated by some grammar/recognized by some machine
$\mathfrak{L}(G)$	The class of languages recognized by G-automata
$\mathfrak{L}(G)^{s}_{t(n)}$	The class of languages recognized by strongly $t(n)$-time bounded G-automata
$\mathfrak{L}(G)^{w}_{t(n)}$	The class of languages recognized by weakly $t(n)$-time bounded G-automata
$\mathfrak{L}(\mathcal{M})$	The class of languages recognized by the machine type $\mathcal{M}$
$\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M)$	The class of languages generated by context-free valence grammars over M
$\mathfrak{L}(\mathrm{Val}, \mathrm{NFA}, M)$	The class of languages recognized by valence automata over M
$\mathfrak{L}(\mathrm{Val}, \mathrm{PDA}, M)$	The class of languages recognized by valence pushdown automata over M
$\mathfrak{L}(\mathrm{Val}, \mathrm{REG}, M)$	The class of languages generated by regular valence grammars over M
M	The set of matrices multiplied with the vector of a homing vector automaton
M	A monoid
$M_n(\mathbb{Z})$	The set of $n \times n$ matrices with integer entries
$\mathbb{N}$	The set of natural numbers
$o(.)$	Little-o notation
$O(.)$	Big-O notation
$\mathfrak{P}$	A valence pushdown automaton
$\mathcal{P}(A)$	The power set of a set A
P_2	The polycyclic monoid of rank 2
$P(X)$	The polycyclic monoid on X
$\mathbb{Q}$	The set of rational numbers
$\mathbb{Q}^+$	The set/group of nonzero rational numbers
Q	The set of states
q_1	The initial state
Q_a	The set of accept states
S	A semigroup
$S/\mathtt{I}$	The Rees quotient semigroup

$S_k(m)$	The set of $k \times k$ matrices whose entries belong to the set $\{-m, \ldots, m\}$ for some positive integer m
$SL(n, \mathbb{Q})$	The special linear group of order n over the field of rationals
$SL(n, \mathbb{Z})$	The special linear group of order n over the field of integers
$\mathcal{T}$	A Turing machine
$U_L(n)$	The maximum k such that there exist k distinct strings that are uniformly n-dissimilar for L
v	A vector
$\mathcal{V}$	A vector automaton
$v[i]$	The i'th entry of the vector v
$\mathcal{W}$	A finite automaton with multiplication
$\|w\|$	The length of the string w
$W(G)$	The word problem language of the group G
$\|w\|_\sigma$	The number of occurrences of σ in w
$w[i]$	The i'th symbol of the string w
w^r	The reverse of the string w
$\mathbb{Z}$	The set of integers
$\mathbb{Z}^k$	The set/group of k-dimensional integer vectors
δ	A transition function
ε	The empty string
Γ	A tape/stack alphabet
Γ_ε	$\Gamma \cup \{\varepsilon\}$
Λ	The set of multipliers of a finite automaton with multiplication
ω	A register status
Ω	The set of register status
$\phi(L)$	The Parikh image of a language L
$\phi(w)$	The Parikh image of a string w
σ	A symbol
Σ	An alphabet
Σ^*	The set of all strings over Σ

$\Sigma_{\$}$	$\Sigma \cup \{\$\}$
Σ_{ε}	$\Sigma \cup \{\varepsilon\}$
θ	A counter status
Θ	The set of counter status
$\simeq$	Isomorphic
$\downarrow$	Stay
$\rightarrow$	Right
$\leftarrow$	Left
$\sqcup$	Blank symbol
\$	End-marker

LIST OF ACRONYMS/ABBREVIATIONS

0-1DFAMW	Stateless one-way deterministic finite automaton with multiplication without equality
0-DBHVA(k)	Stateless real-time k-dimensional deterministic blind homing vector automaton
0-DFAMW	Stateless real-time deterministic finite automaton with multiplication without equality
0-DHVA(k)	Stateless real-time k-dimensional deterministic homing vector automaton
0-NBHVA(k)	Stateless real-time k-dimensional nondeterministic blind homing vector automaton
0-NFAMW	Stateless real-time nondeterministic finite automaton with multiplication without equality
0-NHVA(k)	Stateless real-time k-dimensional nondeterministic homing vector automaton
1DBHVA(k)	One-way k-dimensional deterministic blind homing vector automaton
1DFA	One-way deterministic finite automaton
1DFAM	One-way deterministic finite automaton with multiplication
1DFAMW	One-way deterministic finite automaton with multiplication without equality
1DHVA(k)	One-way k-dimensional deterministic homing vector automaton
1DkBCA	One-way deterministic blind k-counter automaton
1DkCA	One-way deterministic k-counter automaton
1NBHVA(k)	One-way k-dimensional nondeterministic blind homing vector automaton
1NHVA(k)	One-way k-dimensional nondeterministic homing vector automaton
1NFA	One-way nondeterministic finite automaton
1NFAM	One-way nondeterministic finite automaton with multiplication

1NFAMW	One-way nondeterministic finite automaton with multiplication without equality
1NkBCA	One-way nondeterministic blind k-counter automaton
1NkCA	One-way nondeterministic k-counter automaton
1NPDA	One-way nondeterministic pushdown automaton
DBHVA(k)	Real-time k-dimensional deterministic blind homing vector automaton
DFA	Real-time deterministic finite automaton
DFAM	Real-time deterministic finite automaton with multiplication
DFAMW	Real-time deterministic finite automaton with multiplication without equality
DHVA(k)	Real-time k-dimensional deterministic homing vector automaton
DkBCA	Real-time deterministic blind k-counter automaton
DkCA	Real-time deterministic k-counter automaton
DVA(k)	Real-time k-dimensional deterministic vector automaton
CA	Counter automaton
CF	The class of context-free languages
FA	Finite automaton
FAM	Finite automaton with multiplication
HVA(k)	k-dimensional homing vector automaton
kBCA	blind k-counter automaton
kCA	k-counter automaton
NBHVA(k)	Real-time k-dimensional nondeterministic blind homing vector automaton
NHVA(k)	Real-time k-dimensional nondeterministic homing vector automaton
NkBCA	Real-time nondeterministic blind k-counter automaton
NkCA	Real-time nondeterministic k-counter automaton
NFA	Real-time nondeterministic finite automaton
NFAM	Real-time nondeterministic finite automaton with multiplication

NFAMW	Real-time nondeterministic finite automaton with multiplication without equality
PDA	Pushdown automaton
RE	The class of recursively enumerable languages
REG	The class of regular languages
TISP$(t(n), s(n)$	The class of languages that can be decided by a deterministic Turing machine within $t(n)$ time and $s(n)$ space where n is the length of the input
TM	Turing machine

1. INTRODUCTION

1.1. Automata Theory

The theory of computation aims to investigate the computation process and tries to answer the question "What are the fundamental capabilities and limitations of computers?" as stated by Sipser [1]. To study the computation process, we have to first formalize the notions of computational problems and computational models.

At the heart of computational problems, lie the decision problems, the problems whose answers are either yes or no. Any decision problem can be represented by the set of instances which have the answer yes. Consider the problem of checking whether a number is prime. This problem can be represented by the set $\{2, 3, 5, 7, \dots\}$, where the members of the set are the prime numbers. Furthermore, the elements of the set can be represented as *strings*, a finite sequence of *symbols* belonging to a finite set called the *alphabet*. For instance, the set of prime numbers can be expressed by the set $\{11, 111, 11111, 1111111, \dots\}$, where the alphabet contains the single symbol 1.

Automata theory is the study of computational models that solve decision problems. An automaton is as an abstract machine which processes an input string and makes the decision of yes or no, more technically called as acceptance or rejection. Automata allow us to formalize the notion of computation and serve as mathematical models for computing devices. The set of all accepted input strings is called the *language* recognized by the machine. If there exists a machine whose language is the set of yes instances of a problem, than we have a machine solving the problem.

The most basic model which is known as the *finite automaton* or *finite state machine* is an abstract model for computation with finite memory. It is assumed that the input string is written on a tape and the machine has a tape-head reading the string from left to right. There exist finitely many states and a set of rules governing the transitions between these states. Computation starts from a designated initial

state and one input symbol is consumed at each step. An input string is accepted if after reading the string, computation ends in a special state called the accept state and otherwise rejected.

One of the founders of the theory of formal languages, Noam Chomsky, defined four classes of languages, namely the classes of regular languages, context-free languages, context-sensitive languages and recursively enumerable languages, forming the Chomsky Hierarchy. Finite automata recognize exactly the class of regular languages.

1.2. Extended Models of Finite Automata

Throughout the literature, a variety of automaton models has been proposed. Many different models of automata that have been examined can be regarded as a finite automaton augmented with some additional memory. The type of the memory, restrictions on how this memory is accessed, computation mode and the conditions for acceptance determine the expressiveness of the model in terms of language recognition. One can list pushdown automata [2], counter machines [3] and Turing machines [4] among the many such proposed models.

Pushdown automata are finite automata augmented with a stack, a memory which can be used in the last-in-first-out manner. Their *nondeterministic* variants, in which there may be more than one possible move at each step, and the acceptance condition is the existence of at least one computational path that ends in an accept state, recognize exactly the class of context-free languages. Note that a language is context-free if it is generated by a *context-free grammar*, which is a collection consisting of a set of variables and terminals, and a set of production rules. Starting from the start variable, the rules describe how to generate a string of terminals, by replacing each variable with a string of variables and terminals. The set of all generated strings is the language of the grammar.

The foundations of the theory of computation have been established by Alan Turing, who proposed the *Turing machine* as a model for universal computation [4].

A Turing machine is a finite automaton equipped with an infinite read-write tape which is allowed to move in both directions. The Turing machine is a model for our computers and for the computation process performed by the human mind. A language is recognized by a Turing machine if for every string in the language, the computation on the string ends in the accept state. The class of languages recognized by Turing machines is known as the *recursively enumerable* or *Turing recognizable* languages.

Counter machines are finite automata equipped with additional registers that are initialized to zero at the beginning and can be incremented or decremented based on the current state and the status of the counters (zero or nonzero) throughout the computation. A finite automaton with 2 counters is as powerful as a Turing machine, which led researchers to add various restrictions to the definition. For instance, in a *blind counter automaton*, the counters cannot be checked until the end of the computation and the next move depends only on the current state and the scanned symbol. The class of languages recognized by nondeterministic blind counter automata are incomparable with the class of context-free languages.

Another variant is the *extended finite automaton* (finite automaton over a group, group automaton, G-automaton), which is a nondeterministic finite automaton equipped with a register that holds an element from a group [5]. The register is initialized to the identity element of the group, and a computation is deemed successful if the register is equal to the identity element at the end of the computation after being multiplied with a group element at every step. The computational power of a G-automaton is determined by the group G. This setup generalizes various models such as pushdown automata, Turing machines, nondeterministic blind counter automata and finite automata with multiplication [6]. When a monoid is used instead of a group, then the model is also called monoid automaton or M-automaton. The same model also appears under the name of valence automata in various papers.

The notion of extended finite automata is also strictly related to that of valence grammar introduced by Păun in [7]. A *valence grammar* is a formal grammar in which every rule of the grammar is equipped with an element of a monoid called

the valence of the rule. Words generated by the grammar are defined by successful derivations. A successful derivation is a derivation that starts from the start symbol of the grammar such that the product of the valences of its productions (taken in the obvious order) is the identity of the monoid. Valence pushdown automata, which are pushdown automata equipped with a register that is multiplied by elements from a monoid at each step, and context-free valence grammars are discussed in [8].

We have also introduced a new model called vector automaton in [9]. A *vector automaton* is a finite automaton which is endowed with a vector, and which can multiply this vector with an appropriate matrix at each step. One of the entries of this vector can be tested for equality to a rational number. The machine accepts an input string if the computation ends in an accept state, and the test for equivalence succeeds.

In order to incorporate the notion of the computation being successful if the register returns to its initial value at the end of the computation as in the case of extended finite automata to this setup, we propose the new *homing vector automaton* (HVA) model. A homing vector automaton can multiply its vector with an appropriate matrix at each step and can check the entire vector for equivalence to the initial value of the vector. The acceptance criterion is ending up in an accept state with the value of the vector being equal to the initial vector.

1.3. Contributions and Overview

The aim of this thesis is to investigate extended models of finite automata focusing mainly on finite automata over groups and homing vector automata. We examine the classes of languages that can be recognized by different variants of these models and compare them with the classes of languages recognized by the classical models. We prove separation results based on the different restrictions imposed on the models.

Much of the current literature on extended finite automata pays particular attention to finite automata over free groups and free Abelian groups. This study makes a major contribution to the research on extended finite automata by exploring finite

automata over matrix groups for the first time. Most of these results were published in [10–12].

Matrices play an important role in many areas of computation and many important models of probabilistic and quantum computation [13, 14] can be viewed in terms of vectors being multiplied by matrices. The motivation behind analyzing homing vector automata is the matrix multiplication view of programming, which abstracts the remaining features of such models away. We investigate homing vector automata under several different regimes, which helps us to determine whether different parameters confer any additional recognition power. Our results on homing vector automata have previously appeared in [15–17].

We also present some results on context-free valence grammars and valence pushdown automata. These results are mainly some generalizations of the previously established results for the theory of extended finite automata and appeared in [18].

The rest of the thesis is structured as follows:

Chapter 2 contains definitions of basic terminology and formal definitions of some of the classical models. It provides a framework for the rest of the thesis. A background on algebra is also presented.

In Chapter 3, we investigate the language classes recognized by finite automata over matrix groups. For the case of 2×2 matrices, we prove that the corresponding finite automata over rational matrix groups are more powerful than the corresponding finite automata over integer matrix groups. Finite automata over some special matrix groups, such as the discrete Heisenberg group and the Baumslag-Solitar group are also examined. We also introduce the notion of time complexity for group automata and demonstrate some separations among related classes. The case of linear-time bounds is examined in detail throughout our repertory of matrix group automata. Furthermore, we look at the connection between decision problems for matrix groups and finite automata over matrix groups.

Chapter 4 defines the homing vector automaton and introduces the various limited versions that we will use to examine the nature of the contribution of different aspects of the definition to the power of the machine. A generalized version of the Stern-Brocot encoding method, suitable for representing strings on arbitrary alphabets, is developed. The computational power and properties of deterministic, nondeterministic, blind, non-blind, real-time and one-way versions of these machines are examined and compared to various related types of automata. We establish a connection between one-way nondeterministic version of homing vector automata and extended finite automata. As one-way versions are too powerful even in the case of low dimensions, we pay special attention to real-time homing vector automata. Some closure properties of real-time homing vector automata and their stateless (one state) versions are investigated.

In Chapter 5, we focus on pushdown valence automata and context-free valence grammars. We investigate valence pushdown automata, and prove that they are only as powerful as valence automata. We observe that certain results proven for monoid automata can be easily lifted to the case of context-free valence grammars.

Chapter 6 is the conclusion of the thesis. We summarize the results and list some open questions which form a basis for future research.

2. BACKGROUND

2.1. Basic Notation and Terminology

2.1.1. Sets

Let A be a set. $|A|$ denotes the cardinality of A. The power set of A is denoted by $\mathcal{P}(A)$. A subset $A \subseteq \mathbb{N}^n$ is a *linear* set if there exist vectors $v_0, v_1, \ldots, v_k \in \mathbb{N}^n$ such that

$$A = \{v | v = v_0 + \Sigma_{i=1}^{k} c_i v_i, c_i \in \mathbb{N}\}.$$

A *semilinear* set is a finite union of linear sets.

The *Cartesian product of sets* $A_1, A_2, \ldots, A_n$ is the set of all ordered n-tuples $(a_1, a_2, \ldots, a_n)$ where $a_i \in A_i$ for $i = 1, 2, \ldots, n$. The Cartesian product is denoted by either $A_1 \times A_2 \times \cdots \times A_n$ or by $\prod_{i=1}^{n} A_i$.

2.1.2. Strings and Languages

An *alphabet* is a finite set of symbols and usually it is denoted by Σ. A *string* (*word*) over Σ is obtained by concatenating zero or more symbols from Σ. The string of length zero is called the *empty string* and denoted by ε. The set $\Sigma \cup \{\varepsilon\}$ is denoted by Σ_ε in short. We denote by Σ^* the set of all words over Σ.

For a string $w \in \Sigma^*$, w^r denotes its reverse, $|w|$ denotes its length, $w[i]$ denotes its i'th symbol, $|w|_\sigma$ denotes the number of occurrences of $\sigma \in \Sigma$ in w.

A *language* $L \subseteq \Sigma^*$ is a set of strings over Σ. For a given language L, its complement is denoted by $\bar{L}$. For a given string w, L_w denotes the singleton language containing only w.

For a string $w \in \Sigma^*$ where $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ is an ordered alphabet, the *Parikh image of* w is defined as

$$\phi(w) = (|w|_{\sigma_1}, |w|_{\sigma_2}, \ldots, |w|_{\sigma_k}).$$

For a language L, its Parikh image is defined as

$$\phi(L) = \{\phi(w) | w \in L\}.$$

A language is called *semilinear* if $\phi(L)$ is semilinear.

A language $L \subseteq \Sigma^*$ is said to be *bounded* if there exist words $w_1, \ldots, w_n \in \Sigma^+$ such that $L \subseteq w_1^* \cdots w_n^*$. A bounded language is said to be (*bounded*) *semilinear* if there exists a semilinear set A of $\mathbb{N}^n$ such that

$$L = \{w_1^{a_1} \cdots w_n^{a_n} : (a_1, \ldots, a_n) \in A.\}$$

2.1.3. Vectors and Matrices

For a given row vector v, $v[i]$ denotes its i'th entry. Let $A_{k \times l}$ be a $k \times l$ dimensional matrix. $A[i, j]$ denotes the entry in the i'th row and j'th column of A.

The *identity matrix* I of size n is the $n \times n$ matrix with ones on the main diagonal and zeros elsewhere.

2.2. Automata and Computation

In this section, we are going to talk about basic notions of computation. We refer to the finite automaton model which may be extended with a storage mechanism as discussed in Section 1.1 and Section 1.2.

For a machine $\mathcal{M}$, there exist a finite set of *states* $Q = \{q_1, \ldots, q_n\}$ where q_1 is the *initial state* unless otherwise specified and a set of *accept state(s)* $Q_a \subseteq Q$. An input string $w \in \Sigma^*$ is written on a one-way infinite tape starting from the leftmost tape square. The transition function denoted by δ describes the next move of $\mathcal{M}$ upon reading some input symbol. Initially, the tape-head is placed on the leftmost tape square. Starting from the initial state, the sequence of transitions performed by $\mathcal{M}$ on any input string is called a *computation*.

The acceptance criteria of an input string depend on the type of the machine which we will discuss in detail in Section 2.3. A computation is called *accepting* if the computation results in the acceptance of the string, and *rejecting* otherwise. The set of all strings accepted by $\mathcal{M}$ is called the *language recognized by* $\mathcal{M}$ and denoted by $L(\mathcal{M})$.

2.2.1. Deterministic Computation

A computation is *deterministic* if there is only one possible move at each step. When an input string is read, there is only a single computation.

2.2.2. Nondeterministic Computation

In a *nondeterministic* computation, there may be more than one possible move at each step. When an input string is read, the computation looks like a tree since there may be more than one computation path.

2.2.3. Blind Computation

For the machine types which have additional storage mechanisms like registers or counters, computation is called *blind* if the status of the storage mechanism cannot be checked until the end of the computation. The next move of the machine is not affected by the current status of the storage mechanism. When the computation ends, the status of the storage mechanism is checked and it determines whether the input

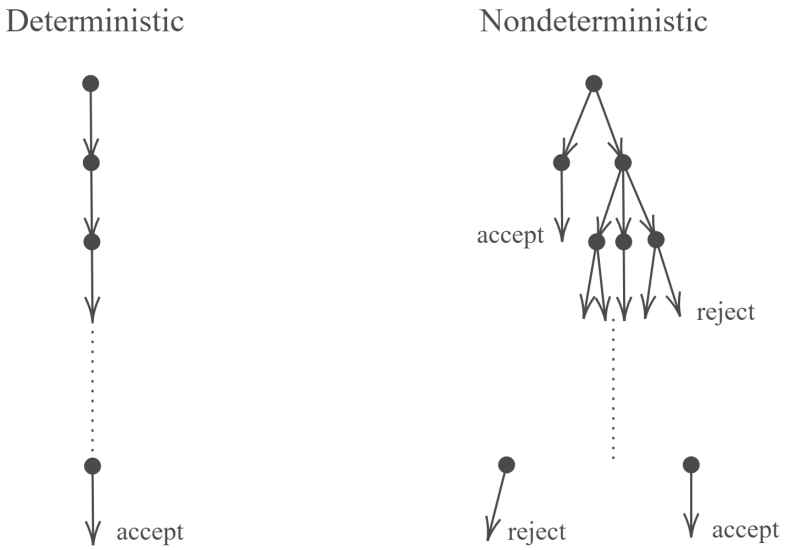


Figure 2.1. Deterministic and nondeterministic computation

string will be accepted or not.

2.2.4. Empty String Transitions

When a machine makes an *empty string* (ε) transition, it moves without consuming any input symbol. In deterministic machines, ε moves should be defined carefully as they may lead to nondeterminism.

2.2.5. Real-time, One-way and Two-way Computation

A computation is called *real-time* if the tape-head moves right at each step. A computation is called *one-way* if the tape-head is allowed to stay on the input tape while moving from left to right. This can be accomplished by adding an additional direction component to the transition function which dictates the movement of the tape-head. A computation is called two-way if the tape-head can move both left and right. Tape head directions will be specified by a subset of the set $\mathcal{D} = \{\leftarrow, \downarrow, \rightarrow\}$ where $\leftarrow, \downarrow$, and $\rightarrow$ stand for left, stay and right respectively.

Note that a machine making ε-transitions does not operate in real-time. A one-way nondeterministic computation may be also defined without specifying the head directions but allowing ε-moves instead. In that case, it is assumed that the machine moves right as long as it consumes an input symbol.

2.2.6. End-marker

In some models, the input string is written on the tape in the form $w\$$ and the machine is allowed to make transition(s) after finishing reading the input string, upon scanning the end-marker \$, which we call postprocessing. Postprocessing may add additional power depending on the model.

2.3. Classical Models

2.3.1. Finite Automaton

A *finite automaton* (FA) is a 5-tuple

$$\mathcal{F} = (Q, \Sigma, \delta, q_1, Q_a),$$

where Q is the set of *states*, Σ is the *input alphabet*, δ is the *transition function*, $q_1 \in Q$ is the *initial state* and $Q_a \subseteq Q$ is the set of *accept states*. The transitions of $\mathcal{F}$ depend only on the current state and the input symbol.

Formally, the transition function of a one-way deterministic finite automaton (1DFA) is defined as follows:

$$\delta : Q \times \Sigma \to Q \times \mathcal{D},$$

where $\mathcal{D} = \{\downarrow, \rightarrow\}$ is the set representing the possible moves of the tape-head, $\downarrow$ denoting stay and $\rightarrow$ denoting right.

$\delta(q, \sigma) = (q', d)$ means that $\mathcal{F}$ moves to state $q' \in Q$ moving its tape-head in direction $d \in \mathcal{D}$ upon reading $\sigma \in \Sigma$ in state $q \in Q$. We assume that the last move of the machine always moves the tape-head right.

In a *real-time deterministic finite automaton* (DFA), the tape-head moves right at every step and the direction component for the tape-head is omitted from the transition function:

$$\delta : Q \times \Sigma \to Q.$$

Let us define the nondeterministic variants of finite automata. The transition function of a *one-way nondeterministic finite automaton* (1NFA) is defined as

$$\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q),$$

so that there may be more than one possible move at each step and the machine is allowed to make ε-transitions.

A *real-time nondeterministic finite automaton* (NFA) is not allowed to perform any ε-transitions. The transition function of an NFA is defined as follows:

$$\delta : Q \times \Sigma \to \mathcal{P}(Q).$$

An input string w of length n is accepted by a finite automaton if there is a computation in which the machine enters an accept state with the tape-head on the $n + 1$'st tape square.

1NFAs, 1DFAs, NFAs and DFAs recognize the same class of languages known as the class of *regular* languages, abbreviated by REG.

2.3.2. Pushdown Automaton

A *pushdown automaton* (PDA) is a finite automaton equipped with a stack. Stacks are one-way infinite storage mechanisms working in last-in-first-out fashion. The operations applied on a stack are called the *pop* and *push* operations, which stand for removing the topmost symbol from the stack and adding a new symbol on top of the stack by pushing down the other symbols in the stack, respectively. Note that the stack alphabet may be different than the input alphabet. At each step of the computation, the machine may pop the topmost symbol from the stack, move to a new state, and push a new symbol onto the stack, depending on the current state and the input symbol.

Formally, *a one-way nondeterministic pushdown automaton* (1NPDA) is a 6-tuple

$$\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_1, Q_a),$$

where Γ is the stack alphabet. The transition function of a 1NPDA is defined as

$$Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon),$$

where $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$. $(q_2, \gamma_2) \in \delta(q_1, \sigma, \gamma_1)$ means when in state q_1 reading $\sigma \in \Sigma_\varepsilon$, $\mathcal{A}$ pops $\gamma_1 \in \Gamma_\varepsilon$ from the stack, moves to state q_2 and pushes $\gamma_2 \in \Gamma$ onto the stack. Note that γ_1 and γ_2 can be ε in which case nothing is popped from or pushed onto the stack. If γ_1 is not on top of the stack, then the transition cannot take place.

A string of length n is accepted by a 1NPDA if there exists a computation in which the machine enters an accept state with the tape-head on the $n+1$'st tape square and the stack is empty. There are also alternative definitions for acceptance which do not require an empty stack. By using ε-transitions, it is easy to see that the stack may be emptied in an accept state to satisfy the additional empty stack requirement and the two definitions correspond to the same class of languages, as long as the computation is one-way and ε-moves are possible.

1NPDAs recognize the class of context-free (CF) languages.

2.3.3. Turing Machine

A *Turing machine* is a finite state automaton with the following properties:

- The tape-head can move in both directions.
- The tape-head can read from the tape and modify the tape content by writing on the tape.

At the beginning of the computation, the input is written on the tape starting from the first tape square and the rest of the tape contains the special blank symbol. If the tape-head tries to move left on the leftmost square, its position does not change.

Formally, a (two-way deterministic) *Turing machine* (TM) is a 7-tuple

$$\mathcal{T} = (Q, \Sigma, \Gamma, \delta, q_1, q_a, q_r),$$

where Σ is the input alphabet not containing the blank symbol $\sqcup$, Γ is the tape alphabet where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$, $q_a \in Q$ and $q_r \in Q$ are the accept and reject states respectively. The transition function of a TM is defined as follows:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \mathcal{D}$$

where $\mathcal{D} = \{\leftarrow, \rightarrow\}$, meaning that $\mathcal{T}$ moves to state $q' \in Q$, updating the tape square under the tape-head by $\gamma_2 \in \Gamma$, moving the tape-head in the direction $d \in \mathcal{D}$, when in state q and reading $\gamma_1 \in \Gamma$ from the tape, specified by the transition $\delta(q, \gamma_1) = (q', \gamma_2, d)$.

While processing an input string, the computation may end at any point (before finishing reading the input string) in the designated accept state resulting in the acceptance of the input string, in the designated reject state resulting in the rejection of the input string or the computation may go on forever without ever entering the

accept state or the reject state.

Turing machines recognize the class of *recursively enumerable* languages (RE).

2.3.4. Counter Automaton

A *counter automaton* (CA) is a finite automaton equipped with one or more counters, a storage mechanism holding an integer which can be incremented, decremented, and checked for equivalence to zero. Formally a CA is a 6-tuple

$$\mathcal{C} = (Q, \Sigma, \delta, q_1, Q_a)$$

and a CA with k counters is abbreviated as kCA. At the beginning of the computation, counters are initialized to 0. At each step of the computation, depending on the current state and the status of the counters, $\mathcal{C}$ moves to another state and updates its counters.
.

The transition function of a *one-way deterministic k-counter automaton* (1DkCA) is defined as follows:

$$\delta : Q \times \Sigma \times \Theta \to Q \times \{-1, 0, 1\}^k \times \mathcal{D}$$

where $\Theta = \{=, \neq\}^k$ and $\mathcal{D} = \{\downarrow, \rightarrow\}$. A transition of the form $\delta(q, \sigma, \theta) = (q', c, d)$ means that upon reading $\sigma \in \Sigma$ in state $q \in Q$, $\mathcal{C}$ moves to q' updating its counters by $c \in \{-1, 0, 1\}^k$ and updates the tape-head with respect to $d \in \mathcal{D}$, given that the status of the counters is $\theta \in \{=, \neq\}^k$, where $=$ and $\neq$ denote whether the corresponding counter values equal zero or not, respectively.

By restricting 1DkCAs so that the status of the counters cannot be checked until the end of the computation, we obtain *one-way deterministic blind k-counter automaton*

(1DkBCA). The transition function of a 1DkBCA is formally defined as follows:

$$\delta : Q \times \Sigma \to Q \times \{-1, 0, 1\}^k \times \mathcal{D}.$$

In a blind counter automaton, the next move of the machine does not depend on the status of the counters.

For the one-way deterministic machines, we assume that the last move of the machine always moves the tape-head to the right.

The real-time versions, *real-time deterministic k-counter automaton* (DkCA) and *real-time deterministic blind k-counter automaton* (DkBCA) are defined analogously, by omitting the direction component from the transition functions. The ranges of the transition functions take the following form: $Q \times \{-1, 0, 1\}^k$.

Let us define the nondeterministic variants of counter automata. The transition function of a *one-way nondeterministic k-counter automaton* (1NkCA) is defined as follows:

$$\delta : Q \times \Sigma_\varepsilon \times \Theta \to \mathcal{P}(Q \times \{-1, 0, 1\}^k).$$

A *one-way nondeterministic blind k-counter automaton* (1NkBCA) is a restricted 1NkCA which cannot check the value of the counters until the end of the computation. Transition function of a 1NkBCA is defined as

$$\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q \times \{-1, 0, 1\}^k).$$

The real-time versions, *real-time nondeterministic k-counter automaton* (NkCA) and *real-time nondeterministic blind k-counter automaton* (NkBCA) are defined analogously by not allowing ε-moves. The domains of the transition functions of these models are replaced with $Q \times \Sigma \times \Theta$ and $Q \times \Sigma$ respectively.

An input string w of length n is accepted by a kCA if there exists a computation in which the machine enters an accept state with the tape-head on the $n+1$'st square. An input string is accepted by a kBCA with the further requirement that all of the counters- values are equal to 0 .

The abbreviations used for counter automata variants discussed so far are given in Table 2.1.

Table 2.1. The abbreviations for CA variants.

	Real-time	One-way
Deterministic	DkCA	1DkCA
Deterministic blind	DkBCA	1DkBCA
Nondeterministic	NkCA	1NkCA
Nondeterministic blind	NkBCA	1NkBCA

A 1D2CA can simulate a Turing Machine [19] and therefore $\bigcup_k \mathfrak{L}(1\mathrm{D}k\mathrm{CA}) = \mathsf{RE}$. As two counters are enough to recognize any recursively enumerable language, a considerable amount of literature has been published on language recognition power of counter automata under various restrictions. Preliminary work on counter automata was undertaken by Fischer *et al.*, who investigated real-time deterministic counter automata [3], and one-way deterministic counter automata [20]. A hierarchy based on the number of the counters for real-time deterministic counter automata is demonstrated in [3]. In [20], the state set is separated into polling states from which the machine moves to another state and updates the counters by consuming an input symbol and autonomous states which allow machine to update the counters and change the state without reading anything. It is easy to show that both definitions are equivalent and correspond to machines with the same language recognition power.

A remarkable result from [20] states the following:

Fact 2.1. [20] *Given any kCA with the ability to alter the contents of each counter independently by any integer between $+c$ and $-c$ in a single step (for some fixed integer c), one can effectively find a time-equivalent (ordinary) kCA.*

The proof involves using some additional states to simulate counter updates from the set $\{-c, -c+1, \ldots, c-1, c\}^k$ with an ordinary counter automaton, without increasing the time complexity and the number of the counters. Hence, we can assume that any kCA can be updated by arbitrary integers at each step.

One-way deterministic and nondeterministic counter automata working under time restrictions formed the central focus of study of Greibach in [21], in which the author proved various separation results between deterministic and nondeterministic models. Let us note that in [21] and [22] where one-way deterministic counter automata are investigated, it is assumed that the counter automata can process the end-marker.

In another major study by Greibach, one-way nondeterministic blind counter automata are examined [23] and some other restricted versions like partially blind counters and reversal bounded counters are introduced as well.

2.3.5. Finite Automata with Multiplication

A *finite automaton with multiplication* (FAM) [6] is 6-tuple

$$\mathcal{W} = (Q, \Sigma, \delta, q_1, Q_a, \Lambda),$$

where the additional component Λ is a finite set of rational numbers (multipliers). A FAM is a finite automaton equipped with a register holding a positive rational number. The register is initialized to 1 at the beginning of the computation and multiplied with a positive rational number at each step, based on the current state, input symbol and the status of the register determined by whether the register is equal to 1 or not. The input string of a FAM is given in the form $w\$$ and FAMs are allowed to perform post-processing.

The original definition of FAMs is given for *one-way* machines where the tape-head is allowed to stay on the same input symbol for more than one step.

A *one-way deterministic finite automaton with multiplication* (1DFAM) is defined by the transition function

$$\delta : Q \times \Sigma_{\$} \times \Omega \to Q \times \mathcal{D} \times \Lambda,$$

where $\Sigma_{\$} = \Sigma \cup \{\$\}$, Ω is the set $\{=, \neq\}$ denoting the whether the register is equal to 1 or not respectively and $\mathcal{D} = \{\downarrow, \rightarrow\}$ is the set of head directions. It is assumed that $\delta(q, \$, \omega) = \emptyset$ for all $q \in Q_a$ so that the computation ends once the end-marker $\$$ is scanned in an accept state. Reading symbol $\sigma \in \Sigma_{\$}$ in state $q \in Q$, $\mathcal{W}$ compares the current value of the register with 1, thereby calculating the corresponding value $\omega \in \Omega$, and switches its state to $q' \in Q$, moves its head in direction $d \in \mathcal{D}$, and multiplies the register by $\lambda \in \Lambda$, in accordance with the transition function value $\delta(q, \sigma, \omega) = (q', d, \lambda)$.

A *one-way deterministic finite automaton with multiplication without equality* (1DFAMW) is a model obtained by restricting 1DFAM so that the register can be checked only at the end of the computation. The transition function of a 1DFAMW is defined as follows:

$$\delta : Q \times \Sigma_{\$} \to Q \times \mathcal{D} \times \Lambda,$$

where the next move of the machine does not depend on the current status of the register. The 1DFAMW can be seen as the blind version of the 1DFAM model.

We define the real-time versions, *real-time deterministic finite automaton with multiplication* (DFAM), and *real-time deterministic finite automaton with multiplication without equality* (DFAMW), by removing the direction component from the transition functions and assuming that the tape-head moves right at each step. The ranges of the transition functions are updated with $Q \times \Lambda$.

A *one-way nondeterministic finite automaton with multiplication* (1NFAM) is a model that extends the 1DFAM with the ability to make nondeterministic moves. The

transition function of a 1NFAM is defined as

$$Q \times \Sigma_{\$} \times \Omega \to \mathcal{P}(Q \times \mathcal{D} \times \Lambda).$$

A *one-way nondeterministic finite automaton with multiplication without equality* (1NFAMW) is the blind version of the 1NFAM model which cannot check whether or not the register has value 1 during computation. Transition function of a 1NFAMW is defined as follows:

$$\delta : Q \times \Sigma_{\$} \to \mathcal{P}(Q \times \mathcal{D} \times \Lambda),$$

so that the next move of the machine does not depend on the current status of the register.

We also define real-time versions *real-time nondeterministic finite automaton with multiplication* (NFAM) and *real-time nondeterministic finite automaton with multiplication without equality* (NFAMW) by removing the direction component from the transition functions and assuming that the tape-head moves right at each step. In that case, the ranges of the transition functions are updated with $\mathcal{P}(Q \times \Lambda)$.

For an input string w of length n, w is accepted by a FAM or a FAMW if there exists a computation in which the machine enters an accept state with the input head on the end-marker \$ and the register is equal to 1.

The abbreviations used for finite automata with multiplication variants discussed so far are given in Table 2.2.

The following characterization of the class of languages recognized by 1NFAMWs for the case where the alphabet is unary is given in [6].

Fact 2.2. [6] *All* 1NFAMW*-recognizable languages over a unary alphabet are regular.*

Table 2.2. The abbreviations for FAM variants.

	Real-time	One-way
Deterministic	DFAM	1DFAM
Deterministic blind	DFAMW	1DFAMW
Nondeterministic	NFAM	1NFAMW
Nondeterministic blind	NFAMW	1NFAMW

Furthermore, bounded languages recognized by 1NFAMWs have also been examined.

Fact 2.3. [6] *A bounded language is recognized by a* 1NFAMW *iff it is semilinear.*

1NFAMWs are defined by the tape-head directions and they process the end-marker. In the next lemma, we show that modifying the definition of 1NFAMW slightly does not change its recognition power.

Lemma 2.4. *Let $\mathcal{W}$ be a* 1NFAMW*. There exists a* 1NFAMW *$\mathcal{W}'$ which does not process the end-marker and defined using ε-transitions that recognizes the same language as $\mathcal{W}$.*

Proof. Given a 1NFAMW $\mathcal{W}_1 = (Q, \Sigma, \delta, q_1, Q_a, \Lambda)$, we construct $\mathcal{W}_2$ from $\mathcal{W}_1$ by first removing the transitions which are traversed upon reading $\sigma \in \Sigma$ and which do not move the tape-head, by using some additional states and ε-transitions as follows: Let Q_s be the set of state-symbol pairs of $\mathcal{W}_1$ such that $(q, \sigma) \in Q_s$ if there is no incoming transition to q that does not move the tape-head and $\delta(q, \sigma) = (q', \lambda, \downarrow)$ for some state $q' \in Q$ and $\lambda \in \Lambda$. For each (q, σ) pair, let $G_{q,\sigma}$ be the graph obtained from the state transition diagram of $\mathcal{W}_1$ by removing all transitions except the ones of the form $\delta(q, \sigma) = (q', \lambda, \downarrow)$. Let $r_{q,\sigma}$ be the subgraph of $G_{q,\sigma}$ induced by the set of reachable vertices from q in $G_{q,\sigma}$. We create a copy of each $r_{q,\sigma}$ which we denote by $r^c_{q,\sigma}$ and connect it to $\mathcal{W}_2$ as follows: From the state q in the original copy, we add an ε-transition to the state q in $r^c_{q,\sigma}$. In $r_{q,\sigma}$, there should be some states t satisfying

$\delta(t, \sigma) = (u, \lambda, \rightarrow)$ for some $u \in Q$ and $\lambda \in \Lambda$, since otherwise the rest of the string cannot be scanned. We remove those transitions from $\mathcal{W}_2$ and add a σ-transition to each original u from the copy of the state t in $r^c_{q,\sigma}$. The inherited transitions from $\mathcal{W}_1$ which do not move the tape-head are removed from $\mathcal{W}_2$.

Next we remove the \$-transitions from $\mathcal{W}_2$. Let $Q_\$$ be the set of states of $\mathcal{W}_1$ that don't have any incoming \$-transition and have an outgoing \$-transition. After finishing reading the string, $\mathcal{W}_1$ should enter a state from $Q_\$$, read the \$ symbol and possibly make some transitions without changing the tape-head and eventually end in an accept state, to accept any string. Let $G_\$$ be the graph obtained from the transition diagram of $\mathcal{W}$, by removing all transitions except the \$-transitions. Let r_q be the subgraph of $G_\$$, induced by the set of reachable vertices from q in $G_\$$, for each $q \in Q_\$$. We create a copy r^c_q of each subgraph r_q, replace the \$ symbols with ε and connect it to $\mathcal{W}_2$: For each incoming transition to q in $\mathcal{W}_1$, we create a copy of the transition and connect it to the copy of q in r^c_q. The \$-transitions inherited from $\mathcal{W}_1$ are removed from $\mathcal{W}_2$ and any accept state of $\mathcal{W}_1$ is no longer an accept state in $\mathcal{W}_2$. $\mathcal{W}_2$ simulates the computation of $\mathcal{W}_1$ on any non-empty string until scanning the \$ and then follows the transitions in the newly added states to reach an accept state.

We can safely remove any remaining tape-head directions which move the tape-head to the right from $\mathcal{W}_2$ and we obtain a 1NFAMW without the tape-head directions and that does not process the end-marker recognizing the same language as $\mathcal{W}_1$. □

From now on, we may assume that a 1NFAMW is defined without the tape-head directions and does not process the end-marker.

2.4. Background on Algebra

In this section we provide definitions for some basic notions from algebra and group theory. See [24, 25] for further references.

2.4.1. Algebraic Structures

Let A be a set. A *binary operation* $*$ on a set A is a function from $A \times A$ to A. Let B be a subset of A. The subset B is *closed under* $*$ if for all $a, b \in B$, we also have $a * b \in B$.

A binary operation $*$ is called

- *associative* if for all $a, b, c \in A$, we have $(a * b) * c = a * (b * c)$,
- *commutative* if $a * b = b * a$ for all $a, b \in A$.

A set A, together with a binary operation $*$ is called an *algebraic structure* $(A, *)$. Let $(A, *)$ and $(A', *')$ be binary algebraic structures. An *isomorphism* of A with A' is a one-to-one function ϕ mapping A onto A' such that $\phi(a * b) = \phi(a) *' \phi(b)$ for all $a, b \in A$. If such a map exists, then A and A' are *isomorphic* binary structures which is denoted by $A \simeq A'$.

For an algebraic structure $(A, *)$,

- an element $e \in A$ is called the *identity element* if for all $a \in A$ $e * a = a * e = a$,
- element $a \in A$ has an *inverse* if there is an element a' in A such that $a * a' = a' * a = e$.

2.4.2. Groups, Monoids, Semigroups

The following are fundamental algebraic structures:

- A *semigroup* $(S, *)$ is an algebraic structure with an associative binary operation.
- A *monoid* $(M, *)$ is a semigroup with an identity element e.
- A *group* $(G, *)$ is a monoid where for every element $a \in G$, there is a unique inverse of $a \in G$ denoted by a^{-1}.

A group may be referred simply as G instead of $(G, *)$ and most of the time the operation sign is omitted and $a * b$ is simply denoted by ab. The *order* $|G|$ of G is the number of elements in G. The *order of an element* g of a group G is the smallest positive integer m such that $g^m = e$. A group is *Abelian* if its binary operation is commutative. A monoid or a semigroup is called *commutative*, if its binary operation is commutative.

A monoid is called *inverse* if for every $x \in M$, there exists a unique $y \in M$ such that $x = xyx$ and $y = yxy$. Note that it is not necessary that xy is equal to the identity of M.

A subset H of a group G is called a *subgroup* of G if

(i) H is closed under the binary operation of G,
(ii) The identity element e of G is in H,
(iii) For all $a \in H$, $a^{-1} \in H$.

A subset H of a monoid M is a *submonoid* if (i) and (ii) hold and a subset H of a semigroup S is a *subsemigroup* if (i) holds.

Let $A \subseteq G$. The *subgroup generated* by A, denoted by $\langle A \rangle$, is the subgroup of G whose elements can be expressed as the finite product of elements from A and their inverses. If this subgroup is all of G, then A *generates* G and the elements of A are called the *generators* of G. If there is a finite set that generates G, then G is *finitely generated*. The smallest cardinality of a generating set for G is the *rank* of the group G. The notion of generating sets also applies to monoids and semigroups.

Let H be a subgroup of a group G. The subset $aH = \{ah | h \in H\}$ of G is the *left coset* of H containing a, and the subset $Ha = \{ha | h \in H\}$ is the *right coset* of H containing a. The number of left cosets of H in G is the *index* of H in G.

Let $G_1, G_2, \ldots, G_n$ be groups. For $(a_1, a_2, \ldots, a_n)$ and $(b_1, b_2, \ldots, b_n)$ in $\prod_{i=1}^{n} G_i = G_1 \times G_2 \times \cdots \times G_n$, define $(a_1, a_2, \ldots, a_n)(b_1, b_2, \ldots, b_n)$ to be the element $(a_1 b_1, a_2 b_2, \ldots, a_n b_n)$. Then $\prod_{i=1}^{n} G_i$ is the *direct product of the groups* G_i under this binary operation. Direct product of monoids and semigroups can be defined similarly.

2.4.3. Groups of Integers, Vectors, Rational Numbers

The set of integers together with the binary operation addition forms a group denoted by $(\mathbb{Z}, +)$. It can be generated by the set $\{1\}$ and its identity element is 0.

The set of k-dimensional integer vectors for some $k \geq 2$ under addition also forms a group denoted by $(\mathbb{Z}^k, +)$ and it is finitely generated by k vectors, i'th vector having 1 in its i'th entry and 0 in the remaining entries for $i = 1 \ldots k$.

The set of positive rationals with the binary operation multiplication forms an infinitely generated group denoted by $(\mathbb{Q}^+, \cdot)$, with the identity element 1.

Note that all groups introduced above are Abelian.

2.4.4. Matrix Groups and Monoids

The set of all $n \times n$ matrices with integer entries with the operation of matrix multiplication forms a monoid, which is denoted by $M_n(\mathbb{Z})$.

We denote by $GL(n, \mathbb{Z})$ *the general linear group of degree* n over the field of integers, that is, the group of $n \times n$ invertible matrices with integer entries. Note that these matrices have determinant ± 1. Restricting the matrices in $GL(n, \mathbb{Z})$ to those that have determinant 1, we obtain the *special linear group of degree* n over the field of integers, $SL(n, \mathbb{Z})$.

Analogously, $GL(n, \mathbb{Q})$ is the group of $n \times n$ invertible matrices with rational entries and $SL(n, \mathbb{Q})$ is the group of $n \times n$ invertible matrices with rational entries

with determinant 1.

2.4.5. Free Monoids and Free Groups

Let $A = \{a_1, a_2, \ldots, a_n\}$ be a finite set. We think of A as an alphabet and its elements a_i as the letters of the alphabet. A *word* is a concatenation of finite elements of A.

The set of all words over A is denoted by A^*. A^* together with the binary operation concatenation is called the *free monoid over* A. The empty word ε, which is obtained by concatenation of zero elements, is the identity element of the free monoid. The set of all nonempty words over A is called the *free semigroup over* A and often denoted by A^+.

Now assume that for every $a_i \in A$, there is a corresponding inverse symbol ${a_i}^{-1}$ and let $A^{-1} = \{{a_1}^{-1}, {a_2}^{-1}, \ldots, {a_n}^{-1}\}$. Consider the set of all words over $X = A \cup A^{-1}$. We can simplify a word by removing occurrences of $a_i {a_i}^{-1}$, for each i. A word is called *reduced* if it cannot be further simplified. The set of all reduced words over X is called the *free group over* A. The number of elements in A is called the *rank of the free group*. An arbitrary group G is called *free*, if it is isomorphic to the free group generated by a subset S of G. Informally, a group is free if no relation holds among the generators of the group. Two free groups are isomorphic iff they have the same rank.

We will denote the *free group* of rank r by $\mathbf{F}_r$. Note that $\mathbf{F}_0$ is the trivial group and $\mathbf{F}_1$ is Abelian and isomorphic to $(\mathbb{Z}, +)$. 2 is the smallest rank of a non-Abelian free group.

The well known Nielsen-Schreier Theorem for free groups states the following.

Fact 2.5. [26, 27] *Every subgroup of a free group is free.*

Furthermore, $\mathbf{F}_2$ contains free subgroups of every finite rank.

2.4.6. Free Abelian Groups

Let A be a subset of a nonzero Abelian group G and suppose that each nonzero element g in G can be expressed uniquely in the form

$$g = n_1 a_1 + n_2 a_2 + \cdots + n_r a_r$$

for $n_i \neq 0$ in $\mathbb{Z}$ and distinct $a_i \in A$. G is called a *free Abelian group* of rank n and A is called a *basis* for the group. Any two free Abelian groups with the same basis are isomorphic.

A free Abelian group of rank r is isomorphic to $\mathbb{Z} \times \mathbb{Z} \times \cdots \times \mathbb{Z} = \mathbb{Z}^r$. Hence, the group of integers $\mathbb{Z}$ and integer vectors $\mathbb{Z}^n$ are finitely generated free Abelian groups. The group of positive rational numbers $\mathbb{Q}^+$ is the free Abelian group of infinite rank.

2.4.7. Word Problem

For any finitely generated group G with the set of generators A, we have a homomorphism $\phi : X^* \to G$ where $X = \{A \cup A^{-1}\}$. Given a group G generated by the set A, the *word problem for group* G is the problem of deciding whether $\phi(w) = 1$ for a given word $w \in X^*$, where 1 is the identity element of G. The *word problem language* of G is the language $W(G, A)$ over X which consists of all words that represent the identity element of G, that is $W(G, A) = \{w \in X^* | \phi(w) = 1\}$. Most of the time, the statements about the word problem are independent of the generating set and the word problem language is denoted by $W(G)$.

3. EXTENDED FINITE AUTOMATA

In this chapter, we investigate extended finite automata over matrix groups. The theory of extended finite automata has been essentially developed in the case of free groups and in the case of free Abelian groups, where strong theorems allow the characterization of the power of such models and the combinatorial properties of the languages recognized by these automata. For groups that are not of the types mentioned above, even in the case of groups of matrices of low dimension, the study of group automata quickly becomes nontrivial, and there are remarkable classes of linear groups for which little is known about the automaton models that they define.

We start with a survey of extended finite automata, and present the basic definitions and observations in Section 3.1.

In Section 3.2, we present several new results about the classes of languages recognized by finite automata over matrix groups. We focus on matrix groups with integer and rational entries. For the case of 2×2 matrices, we prove that the corresponding group automata for rational matrix groups are more powerful than the corresponding group automata for integer matrix groups, which recognize exactly the class of context-free languages. We also explore finite automata over some special matrix groups, such as the discrete Heisenberg group and the Baumslag-Solitar group. The "zoo" of language classes associated with different groups is presented, visualizing known relationships and open problems.

We also introduce the notion of time complexity for group automata, and use this additional dimension to analyze the relationships among the language families recognized by finite automata over various groups. We develop a method for proving that automata over groups where the growth rate of the group and the time are bounded cannot recognize certain languages, even if one uses a very weak definition of time-bounded computation, and use this to demonstrate some new relationships between time-bounded versions of our language classes. The case of linear-time bounds is ex-

amined in detail throughout our repertory of matrix groups. The results are presented in Section 3.3.

In Section 3.4, we make a connection between the membership and identity problems for matrix groups and semigroups and the corresponding extended finite automata. We prove that the decidability of the emptiness and universe problems for extended finite automata are sufficient conditions for the decidability of the subsemigroup membership and identity problems. Using these results, we provide an alternative proof for the decidability of the subsemigroup membership problem for $GL(2,\mathbb{Z})$ and the decidability of the identity problem for $M_2(\mathbb{Z})$. We show that the emptiness and universe problems for $SL(4,\mathbb{Z})$ are undecidable.

3.1. Basic Notions, Definitions and Survey on Extended Finite Automata

This introductory section provides a brief overview of extended finite automata and reviews the literature.

We gave the definition for finite automaton in Section 2.3.1. Now let us look from the point of view of combinatorial group theory.

In [28], a *finite automaton $\mathcal{F}$ over a monoid M* is defined as a finite directed graph whose edges are labeled by elements from M. $\mathcal{F}$ consists of a vertex labeled as the initial vertex and a set of vertices labeled as the terminal vertices such that an element of M is accepted by $\mathcal{F}$ if it is the product of the labels on a path from the initial vertex to a terminal vertex. A subset of M is called *rational* if its elements are accepted by some finite automaton over M. The idea of rational subset of a monoid is introduced for the first time in [29].

When M is a free monoid such as Σ^*, then the accepted elements are words over Σ and the set of accepted words is a language over Σ. If M is finitely generated by a set A, then equivalently a subset of M is called rational if its elements are accepted by some finite automaton over A. By letting A to be a finite alphabet Σ, we see that

the two definitions of finite automaton coincide. Rational subsets of a free monoid are called rational (regular) languages.

An M-automaton recognizing a language over the alphabet Σ can be seen as a finite automaton over the monoid $\Sigma^* \times M$ such that the accepted elements are $(w, 1)$, where $w \in \Sigma^*$. This is stated explicitly in the following proposition by Corson [30]. The proof involves constructing an M-automaton from a finite automaton over $\Sigma^* \times M$ and vice versa.

Fact 3.1. [30] *Let L be a language over an alphabet Σ. Then L is recognized by an M-automaton if and only if there exists a rational subset $R \subseteq \Sigma^* \times M$ such that $L = \{w \in \Sigma^* | (w, 1) \in R\}$.*

Adopting the same definition, one can define a pushdown automaton as a finite state automaton over $\Sigma^* \times M_{cf}$ where M_{cf} is a certain monoid characterizing the context-free languages [28]. A word $w \in \Sigma^*$ is accepted by a pushdown automaton if there is a path from initial vertex to terminal vertex with label $(1, w)$. Replacing M_{cf} with different monoids, it is possible to recognize other classes of languages. This idea coincides with the formal definition of extended finite automaton, which will be discussed next.

The definition of extended finite automata appeared explicitly for the first time in a series of papers by Dassow, Mitrana and Steibe [5, 31, 32]. An extended finite automaton is formally defined as follows:

Let G be a group under the operation denoted by $\circ$ with the neutral element denoted by e. An *extended finite automaton* over the group G is a 6-tuple

$$\mathcal{E} = (Q, \Sigma, G, \delta, q_1, Q_a),$$

where the transition function δ is defined as

$$\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q \times G).$$

$\delta(q, \sigma) \ni (q', g)$ means that when $\mathcal{E}$ reads the symbol (or empty string) $\sigma \in \Sigma_\varepsilon$ in state $q \in Q$, it moves to state $q' \in Q$, and writes $x \circ g$ in the register, where x is the old content of the register and $g \in G$.

The initial value of the register is the neutral element e of the group G. An input string w of length n is accepted if $\mathcal{E}$ enters an accept state with the tape head on the $n + 1$'st square and the content of the register is equal to the identity element of G.

The class of languages recognized by an extended finite automaton over G will be denoted by $\mathfrak{L}(G)$.

An extended finite automaton over G is also called a finite automaton over G or a G-automaton and we will use the three terms interchangeably

Extended finite automata have appeared implicitly throughout the literature as many classical models can be regarded as finite automaton over a particular group. Pushdown automata [2], blind counter automata [23] and finite automata with multiplication without equality [6] are extended finite automata where the group in consideration is the free group, the additive group of integer vectors $\mathbb{Z}^k$, and the multiplicative group of nonzero rational numbers $\mathbb{Q}^+$, respectively.

Mitrana and Stiebe investigate the language recognition power of finite automata over Abelian groups and conclude the following result.

Fact 3.2. [5] *For an Abelian group G, one of the following relations hold:*

$$\mathfrak{L}(G) = \mathsf{REG},$$
$$\mathfrak{L}(G) = \mathfrak{L}(\mathbb{Z}^k), \text{ for some } k,$$
$$\mathfrak{L}(G) = \mathfrak{L}(\mathbb{Q}^+).$$

They also discuss the computational power of deterministic extended finite automata, proving that they are less powerful than their nondeterministic variants. Throughout this thesis, we will focus on nondeterministic extended finite automata.

In the case of the free groups, Dassow and Mitrana observe the following characterization for the classes of context-free languages and recursively enumerable languages. Although it is true that $\mathbf{F}_2$-automata recognize exactly the class of context-free languages, the proof given in [31] is not correct.

Fact 3.3. [31] $\mathfrak{L}(\mathbf{F}_2) = \mathsf{CF}$.

Fact 3.4. [5,32] $\mathfrak{L}(\mathbf{F}_2 \times \mathbf{F}_2) = \mathsf{RE}$.

In 2005, Corson modified the definition of extended finite automaton by allowing the register to be multiplied with monoid elements [30]. Extended finite automaton over a monoid is also called a monoid automaton or M-automaton. Corson focuses on the connection between the word problem of a group G and the set of languages recognized by G-automata. He also provides a proof for the fact that $\mathfrak{L}(\mathbf{F}_2) = \mathsf{CF}$ by extending the work of Gilman [28].

Another line of research on monoid automata was led by Kambites *et al.* The results concerning the class of languages recognized by monoid automata appear in [33–35]. The connection between a given group G and the groups whose word problems recognized by G-automata has been studied in [36–38].

Recent work on the subject includes that of Corson *et al.* [39,40] and Zetzsche [41].

Corson deals with monoid automata recognizing word problems of free products of groups in [39]. Real-time G-automata where ε-transitions are not allowed are analyzed in [40]. Zetzsche investigates the area from a different point of view, not focusing on particular monoids but proving and generalizing various properties of finite automata over a broad class of monoids.

Even though extensive research has been carried out on extended finite automata, no study exists which deal with finite automata over matrix groups. Having presented the main findings on the subject, we will move on to discuss finite automata over matrix groups in the next section.

3.2. Languages Recognized by Finite Automata over Matrix Groups

In this section, we are going to prove some new results about the classes of languages recognized by finite automata over matrix groups. We will start with some observations from the previous studies. In the remaining parts, we will analyze the language recognition power of finite automata over various matrix groups.

3.2.1. Observations

Let us start by noting the following facts, which are true by the definitions of the machines.

- A $\mathbb{Z}^k$-automaton is equivalent to a one-way nondeterministic blind k-counter automaton (1NkBCA).
- A $\mathbb{Q}^+$-automaton is equivalent to a one-way finite automaton with multiplication without equality (1NFAMW).

As mentioned earlier, the characterization of context-free languages by $\mathbf{F}_2$-automata was first stated by Dassow and Mitrana [31], and proven in [30]. Let us recall that $\mathbf{F}_2$ contains any free group of rank $n \geq 2$ [25].

The relation between the classes of languages recognized by free group automata is summarized as follows.

Fact 3.5. [31] $\mathsf{REG} = \mathfrak{L}(\mathbf{F}_0) \subsetneq \mathfrak{L}(\mathbf{F}_1) = \mathfrak{L}(\mathbb{Z}) \subsetneq \mathfrak{L}(\mathbf{F}_2) = \mathsf{CF}$.

The following result states the hierarchy between the classes of languages recognized by $\mathbb{Z}^k$-automata.

Fact 3.6. [42] $\mathfrak{L}(\mathbb{Z}^k) \subsetneq \mathfrak{L}(\mathbb{Z}^{k+1})$ *for* $k \geq 1$.

Let us mention that the class of context-free languages and the class of languages recognized by nondeterministic blind counter automata are incomparable.

Fact 3.7. CF *and* $\mathfrak{L}(\mathbb{Z}^k)$ *are incomparable for all* $k \geq 2$.

Proof. Consider the language $L = \{a^n b^n | n \geq 0\}$ which is a context-free language. Since context-free languages are closed under star, L^* is a context-free language whereas it cannot be recognized by any $\mathbb{Z}^k$-automaton for all $k \geq 1$ by [23]. On the other hand, the non-context-free language $L' = \{a^n b^n c^n | n \geq 0\}$ can be recognized by a $\mathbb{Z}^2$-automaton. □

3.2.2. Automata on Groups of 2×2 and 3×3 Matrices

Let G_2 be the group generated by the matrices

$$M_a = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad M_b = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}.$$

There exists an isomorphism φ from $\mathbf{F}_2$ onto G_2 by [43], meaning that the group G_2 is isomorphic to $\mathbf{F}_2$. Note that M_a and M_b are integer matrices with determinant 1, which proves that $\mathbf{F}_2$ is a subgroup of $SL(2, \mathbb{Z})$.

Now the question is whether $\mathfrak{L}(GL(2,\mathbb{Z}))$ and $\mathfrak{L}(SL(2,\mathbb{Z}))$ correspond to larger classes of languages than the class of context-free languages. We are going to use the following fact to prove that the answer is negative.

Fact 3.8. [30] *Suppose G is a finitely generated group and H is a subgroup of finite index. Then $\mathfrak{L}(G) = \mathfrak{L}(H)$.*

Now we are ready to state our theorem.

Theorem 3.9. $\mathsf{CF} = \mathfrak{L}(\mathbf{F}_2) = \mathfrak{L}(SL(2,\mathbb{Z})) = \mathfrak{L}(GL(2,\mathbb{Z}))$.

Proof. We are going to use Fact 3.8 to prove the result. Since $SL(2,\mathbb{Z})$ has index 2 in $GL(2,\mathbb{Z})$ and $GL(2,\mathbb{Z})$ is finitely generated, $\mathfrak{L}(GL(2,\mathbb{Z})) = \mathfrak{L}(SL(2,\mathbb{Z}))$. Since $\mathbf{F}_2$ has index 12 in $SL(2,\mathbb{Z})$ [44] and $SL(2,\mathbb{Z})$ is finitely generated, $\mathfrak{L}(SL(2,\mathbb{Z})) = \mathfrak{L}(\mathbf{F}_2)$ which is equal to the family of context-free languages by Fact 3.3. □

In the next theorem, we prove that allowing the register to be multiplied by integer matrices whose determinants are not ± 1 does not increase the computational power.

Theorem 3.10. $\mathfrak{L}(M_2(\mathbb{Z})) = \mathsf{CF}$.

Proof. Suppose that an $M_2(\mathbb{Z})$-automaton $\mathcal{E}$ is given. When $\mathcal{E}$ processes an input string, its register is initialized by the identity matrix and multiplied by matrices from $M_2(\mathbb{Z})$. Suppose that in a successful computation leading to acceptance, the register is multiplied by some singular matrix whose determinant is 0. Then the product of the matrices multiplied with the register will have determinant 0 and the register cannot be equal to the identity matrix again. Similarly, if the register is multiplied with a nonsingular matrix whose determinant is not equal to ± 1, then the determinant of the product of the matrices multiplied with the register cannot be equal to 1 again, since $M_2(\mathbb{Z})$ does not contain any matrix with non-integer determinants. Any such edges labeled by a matrix whose determinant is not equal to ± 1 can be removed from

$\mathcal{E}$ to obtain a $GL(2,\mathbb{Z})$-automaton, without changing the accepted language. Since $\mathfrak{L}(GL(2,\mathbb{Z})) = \mathsf{CF}$, the result follows by Theorem 3.9. □

Let us now investigate the group $SL(3,\mathbb{Z})$, the group of 3×3 integer matrices with determinant 1. We start by looking at an important subgroup of $SL(3,\mathbb{Z})$, the discrete Heisenberg group. The discrete Heisenberg group $\mathbf{H}$ is defined as $\langle a, b | ab = bac, ac = ca, bc = cb \rangle$, where $c = a^{-1}b^{-1}ab$ is called the *commutator* of a and b.

$$a = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad c = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Any element $g \in \mathbf{H}$ can be written uniquely as $b^j a^i c^k$.

$$g = \begin{pmatrix} 1 & i & k \\ 0 & 1 & j \\ 0 & 0 & 1 \end{pmatrix} = b^j a^i c^k$$

It is shown in [35] that the languages $\texttt{MULT} = \{x^p y^q z^{pq} | p, q \geq 0\}$, $\texttt{COMPOSITE} = \{x^{pq} | p, q > 1\}$ and $\texttt{MULTIPLE} = \{x^p y^{pn} | p \in \mathbb{N}\}$ can be recognized by $\mathbf{H}$-automata, using the special multiplication property of the group.

Correcting a small error in [35], we rewrite the multiplication property of the elements of $\mathbf{H}$.

$$(b^x a^y c^z)(b^{x'} a^{y'} c^{z'}) = b^{x+x'} a^{y+y'} c^{z+z'+yx'}$$

We can make the following observation using the fact that $\mathfrak{L}(\mathbf{H})$ contains non-context-free languages.

Theorem 3.11. $\mathfrak{L}(SL(2,\mathbb{Z})) \subsetneq \mathfrak{L}(SL(3,\mathbb{Z}))$.

Proof. It is obvious that an $SL(2,\mathbb{Z})$-automaton can be simulated by an $SL(3,\mathbb{Z})$-automaton. Note that $\mathfrak{L}(SL(2,\mathbb{Z}))$ is the family of context-free languages by Theorem 3.9. Since $\mathfrak{L}(\mathbf{H}) \subseteq \mathfrak{L}(SL(3,\mathbb{Z}))$ and the non-context-free language $\texttt{MULT} = \{x^p y^q z^{pq} | p, q \geq 0\}$ can be recognized by an $\mathbf{H}$-automaton [35], the result follows. □

The following result is a direct consequence of Fact 3.8.

Theorem 3.12. $\mathfrak{L}(SL(3,\mathbb{Z})) = \mathfrak{L}(GL(3,\mathbb{Z}))$.

Proof. Since $GL(3,\mathbb{Z})$ is a finitely generated group and $SL(3,\mathbb{Z})$ has finite index in $GL(3,\mathbb{Z})$, the result follows by Fact 3.8. □

We have talked about the discrete Heisenberg group $\mathbf{H}$. Now let us look at a subgroup of $\mathbf{H}$ generated by the matrices B and C, which we will call H_2.

$$B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad C = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$H_2 = \langle B, C | BC = CB \rangle$ is a free Abelian group of rank 2, and therefore it is isomorphic to $\mathbb{Z}^2$.

We conclude the following about the language recognition power of $\mathbb{Z}^2$ and $\mathbf{H}$.

Theorem 3.13. $\mathfrak{L}(\mathbb{Z}^2) \subsetneq \mathfrak{L}(\mathbf{H})$.

Proof. Since $\mathbb{Z}^2$ is a subgroup of $\mathbf{H}$, $\mathfrak{L}(\mathbb{Z}^2) \subseteq \mathfrak{L}(\mathbf{H})$ follows. The inclusion is proper since $\mathbf{H}$-automaton can recognize the language $\texttt{MULT} = \{x^p y^q z^{pq} | p, q \geq 0\}$ [35], whereas any bounded language in $\mathfrak{L}(\mathbb{Q}^+)$ is semilinear by Fact 2.3. □

Now let us move on to the discussion about matrix groups with rational entries. We will start with a special subgroup of $GL(2, \mathbb{Q})$.

For two integers m and n, the *Baumslag-Solitar group* $BS(m, n)$ is defined as $BS(m, n) = \langle a, b | ba^m b^{-1} = a^n \rangle$. We are going to focus on $BS(1, 2) = \langle a, b | bab^{-1} = a^2 \rangle$.

Consider the matrix group G_{BS} generated by the matrices

$$A = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 1/2 & 0 \\ 0 & 1 \end{pmatrix}.$$

Consider the isomorphism $a \mapsto A$, $b \mapsto B$. The matrices A and B satisfy the property $BAB^{-1} = A^2$,

$$\begin{pmatrix} 1/2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -2 & 1 \end{pmatrix},$$

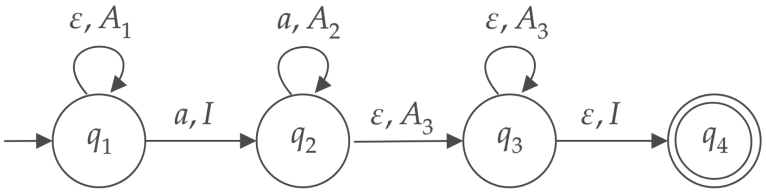
and we conclude that G_{BS} is isomorphic to $BS(1, 2)$.

We will prove that there exists a $BS(1, 2)$-automaton which recognizes a non-context-free language.

Theorem 3.14. $\mathfrak{L}(BS(1, 2)) \nsubseteq \mathsf{CF}$.

Proof. Let us construct a $BS(1, 2)$-automaton $\mathcal{E}$ recognizing the language $\texttt{UPOW} = \{a^{2^n} | n \geq 0\}$. The state diagram of $\mathcal{E}$ and the matrices are given in Figure 3.1. Without scanning any input symbol, $\mathcal{E}$ multiplies its register with the matrix A_1 successively. $\mathcal{E}$ nondeterministically moves to the next state reading the first input symbol without modifying the register. After that point, $\mathcal{E}$ starts reading the string and multiplies its register with the matrix A_2 for each scanned a. At some point, $\mathcal{E}$ nondeterministically stops reading the rest of the string and multiplies its register with the element A_3. After successive multiplications with A_3, $\mathcal{E}$ nondeterministically decides to move to an

accept state.



$$A_1 = \begin{pmatrix} 2 & 0 \\ 1 & 1 \end{pmatrix} \qquad A_2 = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \qquad A_3 = \begin{pmatrix} 1/2 & 0 \\ 0 & 1 \end{pmatrix}$$

Figure 3.1. State transition diagram of $\mathcal{E}$ recognizing UPOW

As a result of i multiplications with A_1, the register has the value

$$\begin{pmatrix} 2^i & 0 \\ 2^i - 1 & 1 \end{pmatrix}$$

before reading the first input symbol. Multiplication with each A_2 leaves 2^i unchanged while subtracting 1 from $2^i - 1$ for each scanned a. The register will have the value

$$\begin{pmatrix} 2^i & 0 \\ 2^i - 1 - j & 1 \end{pmatrix}$$

as a result of j multiplications with the matrix A_2.

For the rest of the computation, $\mathcal{E}$ will multiply its register, say k times, with A_3 resulting in the register value

$$\begin{pmatrix} \frac{2^i}{2^k} & 0 \\ 2^i - 1 - j & 1 \end{pmatrix}$$

since each multiplication with A_3 divides 2^i by 2.

The register contains the identity matrix at the end of the computation if $i = k$ and $j = 2^i - 1$ which is possible if the input string is of the form $a^{1+2^i-1} = a^{2^i}$. In the successful branch, the register will be equal to the identity matrix and $\mathcal{E}$ will end up in the final state having successfully read the input string.

For input strings which are not members of UPOW, either the computation will end before reading the whole input string or the final state will be reached with the register value being different from the identity matrix. Note that $A_1 = B^{-1}A^{-1}$, $A_2 = A$ and $A_3 = B$, where A and B are the generators of the group G_{BS} and recall that G_{BS} is isomorphic to $BS(1,2)$. Since UPOW is a unary nonregular language, it is not context-free and we conclude the result. □

Note that $\mathfrak{L}(\mathbb{Z}) \subsetneq \mathfrak{L}(BS(1,2))$ since the subgroup generated by a in $BS(1,2)$ is isomorphic to $\mathbb{Z}$ and $\mathfrak{L}(BS(1,2))$ contains a unary nonregular language.

We showed that allowing rational entries enlarges the class of languages recognized by 2×2 matrices. What about the group of 2×2 rational matrices with determinant 1? We give a positive answer for the question, by constructing an $SL(2,\mathbb{Q})$-automaton recogizing a unary non-context-free langauge.

Theorem 3.15. $\mathfrak{L}(SL(2,\mathbb{Z})) \subsetneq \mathfrak{L}(SL(2,\mathbb{Q}))$.

Proof. It is obvious that $\mathfrak{L}(SL(2,\mathbb{Z})) \subseteq \mathfrak{L}(SL(2,\mathbb{Q}))$. We will prove that the inclusion is proper.

Let us construct an $SL(2,\mathbb{Q})$-automaton $\mathcal{E}$ recognizing the language $\mathtt{UPOW}_{\mathtt{odd}} = \{a^{2^{2n+1}} \mid n \geq 0\}$. The state diagram of $\mathcal{E}$ and the matrices are given in Figure 3.2. Without scanning any input symbol, $\mathcal{E}$ first multiplies its register with the matrix A_1. $\mathcal{E}$ then multiplies its register with the matrix A_2 successively until nondeterministically moving to the next state. After that point, $\mathcal{E}$ starts reading the string and multiplies its register with the matrix A_3 for each scanned a. At some point, $\mathcal{E}$ nondeterministically stops reading the rest of the string and multiplies its register with the matrix A_4. After

successive multiplications with A_4, $\mathcal{E}$ nondeterministically decides moving to an accept state.

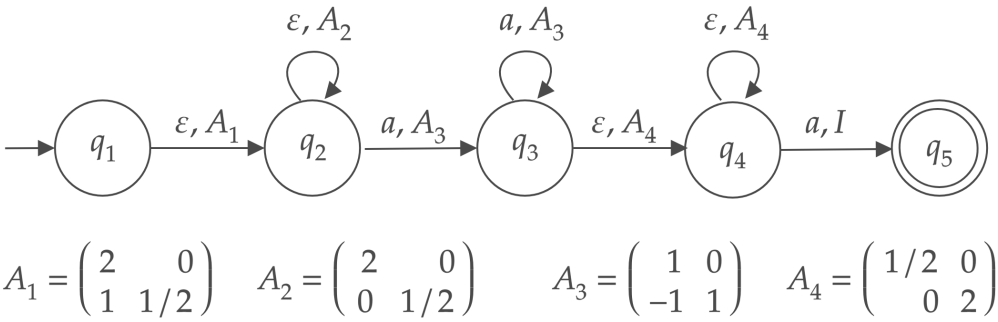


Figure 3.2. State transition diagram of $\mathcal{E}$ recognizing $\texttt{UPOW}_{\texttt{odd}}$

Let us trace the value of the register at different stages of the computation. Before reading the first input symbol, the register has the value

$$\begin{pmatrix} 2^{x+1} & 0 \\ 2^x & \frac{1}{2^{x+1}} \end{pmatrix}$$

as a result of the multiplications with the matrix A_1 and x times the matrix A_2. Multiplication with each A_3 leaves 2^{x+1} and $\frac{1}{2^{x+1}}$ unchanged while subtracting $\frac{1}{2^{x+1}}$ from 2^x for each scanned a. As a result of y multiplications with A_3, the register will have the value

$$\begin{pmatrix} 2^{x+1} & 0 \\ 2^x - \frac{y}{2^{x+1}} & \frac{1}{2^{x+1}} \end{pmatrix}.$$

For the rest of the computation, $\mathcal{E}$ will multiply its register with A_4 until nondeterministically moving to the final state. As a result of z multiplications with A_4, the register will have the value

$$\begin{pmatrix} \frac{2^{x+1}}{2^z} & 0 \\ \left(2^x - \frac{y}{2^{x+1}}\right)\frac{1}{2^z} & \frac{2^z}{2^{x+1}} \end{pmatrix}.$$

The final value of the register is equal to the identity matrix when $y = 2^{2x+1}$ and $z = x + 1$, which is possible only when the length of the input string is 2^{2x+1} for some $x \geq 0$. In the successful branch, the register will be equal to the identity matrix and $\mathcal{E}$ will end up in the final state having successfully read the input string. For input strings which are not members of L, either the computation will end before reading the whole input string, or the final state will be reached with the register value not equaling the identity matrix.

Since the matrices used during the computation are 2 by 2 rational matrices with determinant 1, $L \in \mathfrak{L}(SL(2,\mathbb{Q}))$. $\mathfrak{L}(SL(2,\mathbb{Q}))$ contains a unary nonregular language, which is not true for $\mathfrak{L}(SL(2,\mathbb{Z}))$ by Theorem 3.9 and we conclude the result. □

Let us note that the set of languages recognized by $\mathbb{Q}^+$-automata is a proper subset of the set of languages recognized by $SL(2,\mathbb{Q})$-automata.

Theorem 3.16. $\mathfrak{L}(\mathbb{Q}^+) \subsetneq \mathfrak{L}(SL(2,\mathbb{Q}))$.

Proof. Let $L \in \mathfrak{L}(\mathbb{Q}^+)$ and let $\mathcal{E}$ be a $\mathbb{Q}^+$-automaton recognizing L. We will construct an $SL(2,\mathbb{Q})$-automaton $\mathcal{E}'$ recognizing L. Let $S = \{s_1, \ldots, s_n\}$ be the set of elements multiplied with the register during the computation of $\mathcal{E}$. We define the mapping φ as follows.

$$\varphi : s_i \mapsto \begin{pmatrix} s_i & 0 \\ 0 & \frac{1}{s_i} \end{pmatrix}$$

The elements $\varphi(s_i)$ are 2×2 rational matrices with determinant 1. Let δ and δ' be the transition functions of $\mathcal{E}$ and $\mathcal{E}'$ respectively. We let

$$(q', s_i) \in \delta(q, \sigma) \iff (q', \varphi(s_i)) \in \delta'(q, \sigma)$$

for every $q, q' \in Q$, $\sigma \in \Sigma$ and $s_i \in S$. The resulting $\mathcal{E}'$ recognizes L.

The inclusion is proper since $\texttt{UPOW}_{\texttt{odd}} = \{a^{2^{2n+1}} | n \geq 0\} \in \mathfrak{L}(SL(2,\mathbb{Q}))$ by Theorem 3.15, and $\mathfrak{L}(\mathbb{Q}^+)$ does not contain any unary nonregular language by Fact 2.2, noting that $\mathbb{Q}^+$-automata are equivalent to 1NFAMW's. □

3.2.3. Automata on Matrices of Higher Dimensions

As pointed out in Section 3.1, $\mathbf{F}_2 \times \mathbf{F}_2$-automata are as powerful as Turing machines. Using this fact, we make the following observation.

Theorem 3.17. $\mathsf{RE} = \mathfrak{L}(\mathbf{F}_2 \times \mathbf{F}_2) = \mathfrak{L}(SL(4,\mathbb{Z}))$.

Proof. The first equality is Fact 3.4. Recall from Section 3.2.2 that φ is an isomorphism from $\mathbf{F}_2$ onto G_2, the matrix group generated by the matrices M_a and M_b. Let G' be the following group of matrices

$$\left\{ \begin{pmatrix} & & 0 & 0 \\ M_1 & & & \\ & & 0 & 0 \\ 0 & 0 & & \\ & & & M_2 \\ 0 & 0 & & \end{pmatrix}, \ M_1, \ M_2 \in G_2 \right\}.$$

We will define the mapping $\psi : \mathbf{F}_2 \times \mathbf{F}_2 \to G'$ as $\psi(g_1, g_2) = (\varphi(g_1), \varphi(g_2))$ for all $(g_1, g_2) \in \mathbf{F}_2 \times \mathbf{F}_2$ which is an isomorphism from $\mathbf{F}_2 \times \mathbf{F}_2$ onto G'.

This proves that $\mathbf{F}_2 \times \mathbf{F}_2$ is isomorphic to a subgroup of $SL(4,\mathbb{Z})$. The fact that $\mathfrak{L}(\mathbf{F}_2 \times \mathbf{F}_2)$ is the set of recursively enumerable languages lets us conclude that $\mathfrak{L}(SL(4,\mathbb{Z}))$ is the set of recursively enumerable languages. □

Let us also state that the classes of languages recognized by automata over supergroups of $SL(4,\mathbb{Z})$ such as $GL(4,\mathbb{Z})$ or $SL(4,\mathbb{Q})$ are also identical to the class of recursively enumerable languages.

Theorem 3.18. $\mathfrak{L}(G) = \mathsf{RE}$*, where* G *is any matrix group whose matrix entries are computable numbers and contains* $SL(4,\mathbb{Z})$ *as a subgroup.*

Proof. Note that any finite automaton over a matrix group can be simulated by a nondeterministic Turing machine which keeps track of the register simply by multiplying the matrices and checking whether the identity matrix is reached at the end of the computation, provided that the matrix entries are computable numbers. Since $\mathsf{RE} = \mathfrak{L}(SL(4,\mathbb{Z}))$ and G contains $SL(4,\mathbb{Z})$ as a subgroup, we conclude that $\mathfrak{L}(G)$ is the set of recursively enumerable languages. □

We summarize the results in Figure 3.3. Solid arrows represent proper inclusion, dashed arrows represent inclusion and dashed lines represent incomparability.

3.3. Time Complexity

In the previous section, we compared various automaton models solely on the basis of the groups they employed as a computational resource. The theory of computational complexity deals with various different types of such resources, the allowed runtime of the machines being the most prominent among them. Some of the automata we saw in Section 3.2 (e.g. Figure 3.1) have arbitrarily long computations, and it is a legitimate question to ask whether our results, for instance, the relationships in Figure 3.3, would still hold if one imposed common time bounds on the automata. We study such questions in this section.

3.3.1. Definitions

Before moving on with our discussion, we have to define some new concepts.

A G-automaton $\mathcal{E}$ recognizing language L is said to be *strongly* $t(n)$ *time-bounded* if for any input string x with $|x| = n$, every computation of $\mathcal{E}$ on x takes at most $t(n)$ steps. We will denote the set of languages recognized by strongly $t(n)$-time bounded

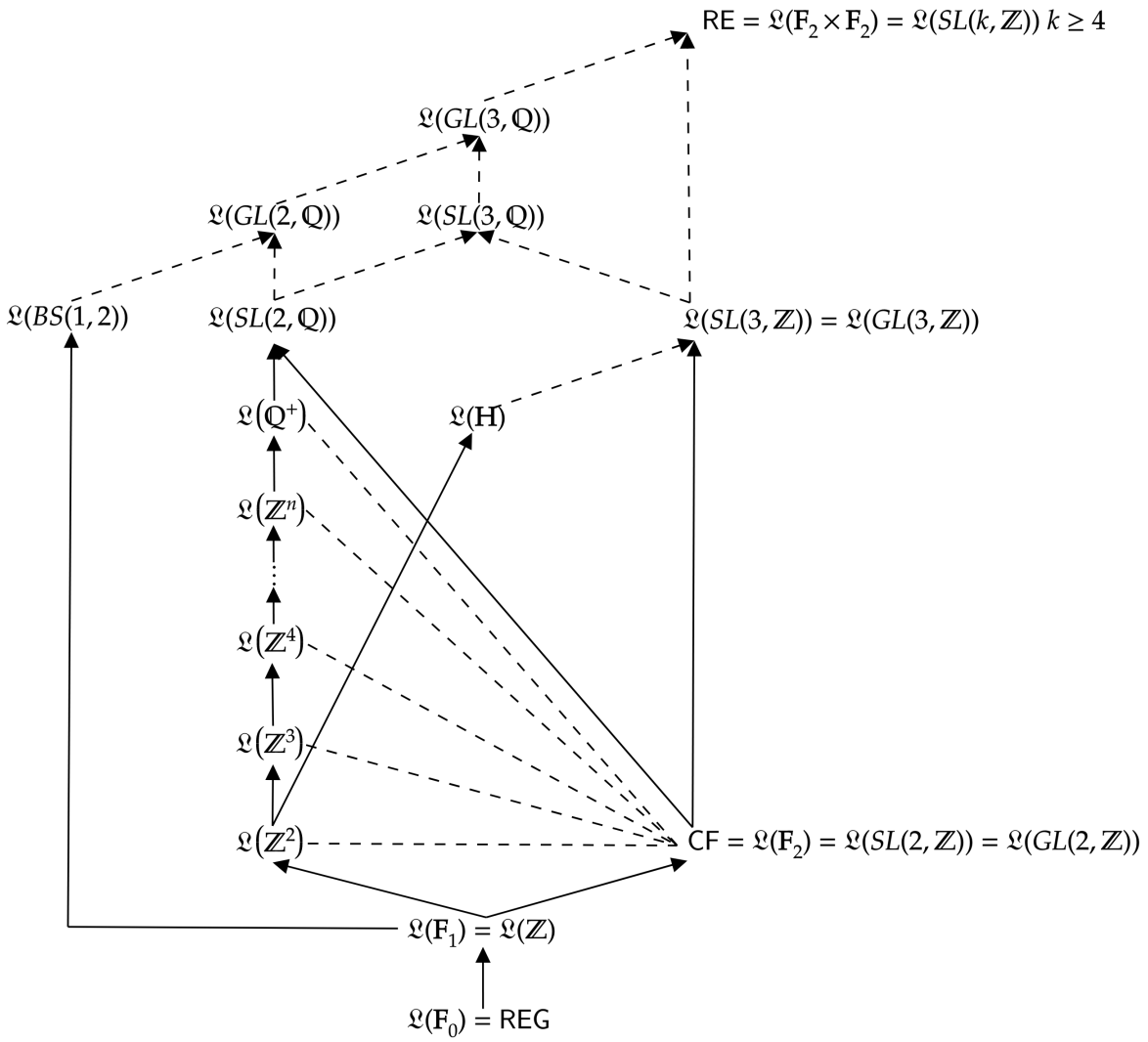


Figure 3.3. Language classes associated with groups

G-automata by $\mathfrak{L}(G)^s_{t(n)}$.

Although the strong mode of recognition defined above is standard in studies of time complexity, we will be able to prove the impossibility results of the next subsection even when the machines are subjected to the following, looser requirement: A G-automaton $\mathcal{E}$ recognizing language L is said to be *weakly* $t(n)$ *time-bounded* if for each accepted input string $x \in L$ with $|x| = n$, $\mathcal{E}$ has a successful computation which takes at most $t(n)$ steps. So any input string is allowed to cause longer computations, as long as none of those are accepting for inputs which are not members of L. We will denote the set of languages recognized by weakly $t(n)$-time bounded G-automata by

$\mathfrak{L}(G)^w_{t(n)}$. Note that the statement $\mathfrak{L}(G)^s_{t(n)} \subseteq \mathfrak{L}(G)^w_{t(n)}$ is true by definition.

Let A be a generator set for the group G. The *length* of $g \in G$, denoted $|g|_A$, is the length of the shortest representative for g in $(A \cup A^{-1})^*$. Let

$$B^A_G(n) = \{g \in G, |g|_A \leq n\}$$

be the set of all elements in G which can be represented by a word of length at most n. The *growth function of a group* G with respect to a generating set A, denoted $g^A_G(n)$, is the cardinality of the set $B^A_G(n)$, that is $g^A_G(n) = |B^A_G(n)|$. The growth function is asymptotically independent of the generating set, and we will denote the growth function of a group G by $g_G(n)$.

For a positive integer n, two strings $w, w' \in \Sigma^*$ are *n-dissimilar for L* if $|w| \leq n$, $|w'| \leq n$, and there exists a string $v \in \Sigma^*$ with $|wv| \leq n$, $|w'v| \leq n$ such that $wv \in L$ iff $w'v \notin L$. Let $A_L(n)$ be the maximum k such that there exist k distinct strings that are pairwise n-dissimilar.

A finite set of strings S is said to be a set of *uniformly n-dissimilar strings for L* if for each string $w \in S$, there exists a string v such that $|wv| \leq n$ and $wv \in L$ and for any string $w' \in S$ such that $w \neq w'$, $|w'v| \leq n$ and $w'v \notin L$. Let $U_L(n)$ be the maximum k such that there exist k distinct strings that are uniformly n-dissimilar.

Note that the following is always true by definition, since the strings in a uniformly n-dissimilar set are pairwise n-dissimilar.

Lemma 3.19. *$U_L(n) \leq A_L(n)$ for all $n \geq 0$.*

3.3.2. Limitations of Machines on Slow Groups Running in Short Time

In this section, we are going to present a method for proving that certain languages cannot be recognized by finite automata over matrix groups when the growth

rate of the group and the time are bounded.

Theorem 3.20. *Let G be a group with growth function $g_G(n)$. $L \notin \mathfrak{L}(G)^w_{t(n)}$ if $g_G(t(n)) \in o(U_L(n))$.*

Proof. Suppose for a contradiction that there exists a weakly $t(n)$ time-bounded G-automaton $\mathcal{E}$ recognizing L in time $t(n)$. For a sufficiently large n, let S be the set of uniformly n-dissimilar strings such that $|S| = U_L(n)$. For every string $w_i \in S$, there exists a string v_i such that $w_i v_i \in L$ and $w_j v_i \notin L$ for all $w_j \in S$ with $i \neq j$.

Let S_{acc} be the set of accepted extended strings of the form $w_i v_i \in L$ with $|w_i v_i| \leq n$ where $w_i \in S$ and $w_j v_i \notin L$ for all $w_j \in S$ with $i \neq j$ and $|w_j v_i| \leq n$. Let C be the set of $t(n)$ time bounded accepting computation paths for the strings in S_{acc}. The computation $c_{w_i v_i} \in C$ on the string $w_i v_i$ can be written as

$$c_{w_i v_i} = c^{w_i}_{w_i v_i} c^{v_i}_{w_i v_i}$$

where $c^{w_i}_{w_i v_i}$ represents the computation up to the end of the prefix w_i and $c^{v_i}_{w_i v_i}$ represents the rest of the computation on the string v_i.

A configuration of a group automaton is a pair consisting of a state and a group element. Let us count the number of configurations that can be reached at the end of the computation $c^{w_i}_{w_i v_i}$. Since the number of states is constant, the number of configurations that can be reached is dependent on the number of different group elements that can appear in the register. After reading a prefix w_i with $|w_i| = m \leq n$, the product of the labels on the edges can be given by $l = g_{i_1} g_{i_2} \dots g_{i_k}$ for some $k \leq t(m)$, since the computation in consideration is time bounded. l can be expressed as a product of κ generators, where κ is at most $C \cdot k$ for some constant C, since each group element labeling a transition in $\mathcal{E}$ is composed of at most some constant number of generators, which is independent of the length of the string. The number of elements in G which can be represented as a product of at most κ generators is given by $g_G(\kappa)$ by the definition of the growth function of G. Hence, the number of different values that can

appear in the register after reading a string of length exactly m is less than or equal to $g_G(\kappa)$. Since $\kappa \leq C \cdot k$ and $k \leq t(m)$ and $g_G(t(n)) \in o(U_L(n))$, we can conclude that

$$g_G(\kappa) \leq g_G(C \cdot t(m)) \in o(U_L(n)).$$

Now it is easy to see that the number of different configurations that can be reached at the end of a computation $c^{w_i}_{w_i v_i}$ is $o(U_L(n))$. Note that the cardinality of the set S, and thus that of S_{acc}, is equal to $U_L(n)$. Due to the pigeonhole principle, the same configuration must be reached at the end of two computations $c^{w_i}_{w_i v_i}$ and $c^{w_j}_{w_j v_j}$ for some $i \neq j$. This will result in the acceptance of the strings $w_i v_j$ and $w_j v_i$, which are not members of L. We arrive at a contradiction and conclude that L cannot be recognized by any weakly $t(n)$ time-bounded G-automaton. □

In the next lemma, we set a lower bound on maximum cardinality of the set of uniformly n-dissimilar strings in the word problem language of some group G.

Lemma 3.21. *Let G be a finitely generated group with growth function $g_G(n)$. Then $U_{W(G)}(n) \geq g_G(\lfloor \frac{n}{2} \rfloor)$.*

Proof. Let A be the generator set of G. The number of distinct elements g in G which can be represented by a word of length less than or equal to $\lfloor \frac{n}{2} \rfloor$ is $g_G(\lfloor \frac{n}{2} \rfloor)$, which is the cardinality of the set $B^A_G(\lfloor \frac{n}{2} \rfloor) = \{g \in G, |g|_A \leq \lfloor \frac{n}{2} \rfloor\}$. Let T be the set containing the string representations of the elements in $B^A_G(\lfloor \frac{n}{2} \rfloor)$. Every $w_i \in T$ can be extended with w_i^{-1} so that the extended string represents the identity element of G and has length less than or equal to n. Since the strings in $W(G)$ are those which belong to $(A \cup A^{-1})^*$ and represent the identity element of G, the extended string $w_i w_i^{-1} \in W(G)$. For every string $w_j \in T$ such that $i \neq j$, $w_j w_i^{-1} \notin W(G)$ since it is not possible for $w_j w_i^{-1}$ to represent the identity element of G. We conclude that the set S is uniformly n-dissimilar. Since $|T| = |B^A_G(\lfloor \frac{n}{2} \rfloor)| = g_G(\lfloor \frac{n}{2} \rfloor)$, it follows that $U_{W(G)}(n) \geq g_G(\lfloor \frac{n}{2} \rfloor)$. □

The following theorem is about the language recognition power of finite automata over polynomial-growth groups which are weakly polynomial time-bounded.

Theorem 3.22. *Let G and H be groups with polynomial and exponential growth functions $g_G(n)$ and $g_H(n)$, respectively. For any polynomial $t(n)$, $\mathfrak{L}(H) \nsubseteq \mathfrak{L}(G)^w_{t(n)}$.*

Proof. Since $U_{W(H)}(n) \geq g_H(\lfloor \frac{n}{2} \rfloor)$ by Lemma 3.21, and $g_H(n)$ is an exponential function, $U_{W(H)}(n)$ is also at least exponential. $g_G(t(n))$ is a polynomial function, since both $g_G(n)$ and $t(n)$ are polynomial. Hence, $W(H) \notin \mathfrak{L}(G)^w_{t(n)}$ by Theorem 3.20, and the result follows since $W(H)$ is trivially in $\mathfrak{L}(H)$. □

Theorem 3.23. *Let G be a group with a polynomial growth function. For any polynomial $t(n)$, $\mathsf{CF} \nsubseteq \mathfrak{L}(G)^w_{t(n)}$.*

Proof. It is known that the word problem of the free group of rank 2, $W(\mathbf{F}_2)$, has an exponential growth function [45]. Assuming that G is a group with polynomial growth function, $W(\mathbf{F}_2)$ cannot be recognized by any weakly $t(n)$ time-bounded G-automaton by Theorem 3.20. Since $W(\mathbf{F}_2)$ is a context-free language, the proof is complete. □

3.3.3. Group Automata Under Linear Time Bounds

Having discussed methods for proving that certain languages can not be recognized by group automata under time restrictions, in this section will we focus on linear-time computation.

Let X be a finite alphabet and let X^* be the free monoid of words over X. For each symbol $x \in X$, let P_x and Q_x be functions from X^* into X^* defined as follows: for every $u \in X^*$,

$$P_x(u) = ux, \quad Q_x(ux) = u.$$

Note that Q_x is a partial function from X^* into X^* whose domain is the language X^*x. The submonoid of the monoid of all partial functions on X^* generated by the set of functions $\{P_x, Q_x \mid x \in X\}$ turns out to be an inverse monoid, denoted by $P(X)$, called the *polycyclic monoid on* X. Polycyclic monoids were explicitly studied by Nivat and Perrot in [46] and have several applications in formal language theory and, in particular, define an interesting storage model of computation for the recognition of formal languages [28, 30, 33, 46, 47].

For any element $x \in X$, $P_xQ_x = 1$ where 1 is the identity element of $P(X)$ and for any two distinct elements $x, y \in X$, P_xQ_y is the empty partial function which represents the zero element of $P(X)$. The partial functions $\{P_x, Q_x\}$ model the operation of pushing and popping x in a PDA, respectively. In order to model popping and pushing the empty string, let us define P_ε and Q_ε as $P_\varepsilon = Q_\varepsilon = 1$. The equivalence between PDA with stack alphabet X and $P(X)$-automata is due to the nature of the functions P_x and Q_x, and investigated in various papers [28, 30, 33]. The resemblance between the free group and $P(X)$ is used to prove that $\mathfrak{L}(\mathbf{F}_2) = \mathsf{CF}$ in [30] and [33].

Our aim is to show that $\mathbf{F}_2$-automata working in linear time can recognize all context-free languages. It is stated in [48] that $P(X)$-automata which consume at least one input symbol at each step are as powerful as $P(X)$-automata without any time bound. However, it is not straightforward to see whether the same is true for $\mathbf{F}_2$-automata.

Theorem 3.24. $\mathfrak{L}(\mathbf{F}_2)^w_{O(n)} = \mathsf{CF}$.

Proof. We are going to use the construction of Kambites [33] to prove that any context-free language can be recognized by a weakly linear-time bounded $\mathbf{F}_2$-automaton.

Let L be a context-free language and let $\mathcal{E} = \{Q, \Sigma, P(X), \delta, q_1, Q_a\}$ be a polycyclic monoid automaton recognizing L. $P(X)$ is the polycyclic monoid on X where the cardinality of the set X is n for some $n \geq 2$. The construction of Kambites provides an $\mathbf{F}_{n+1}$-automaton $\mathcal{E}' = \{Q', \Sigma, \mathbf{F}_{n+1}, \delta', q_1', Q_a'\}$ recognizing the language L. The

generator set for $\mathbf{F}_{n+1}$ is X', where $X' = X \cup \#$.

Let us analyze the construction in more detail.

- $Q' = Q_- \cup Q_+$ where $Q_- = \{q_-|q \in Q\}$ and $Q_+ = \{q_+|q \in Q\}$
- q_1'=q_+ where $q = q_1$.
- $Q_a' = \{q_-|q \in Q_a\}$.
- $\delta'(p_+, \sigma) = (q_+, x\#)$ if $\delta(p, \sigma) = (q, x\#)$ where x is a positive generator for all $\sigma \in \Sigma$.
- $\delta'(p_-, \sigma) = (q_+, x'\#)$ if $\delta(p, \sigma) = (q, x'\#)$ where x' is a negative generator for all $\sigma \in \Sigma$.
- $\delta'(p_+, \sigma) = (q_+, 1)$ if $\delta(p, \sigma) = (q, 1)$ for all $\sigma \in \Sigma$.
- $\delta'(q_+, \varepsilon) = (q_-, 1)$ for each $q \in Q$.
- $\delta'(q_-, \varepsilon) = (q_-, \#^{-1})$ for each $q \in Q$.

We will prove that $\mathcal{E}'$ actually runs in linear time. There are two transitions where the automaton is allowed to move without consuming any input symbols.

For each state $q \in Q$, there are two states q_+ and q_- in $\mathcal{E}'$ which are connected with an edge labeled $(\varepsilon, 1)$. These transitions do not change the register value, and cannot contribute more than half of the runtime of the machine, since at least one input symbol has to be consumed between any two executions of such transitions.

ε-loops exist in the machine $\mathcal{E}'$ for each state q_- where the loop is labeled by $(\varepsilon, \#^{-1})$. Although this looks worrisome at first for the purpose of bounding the runtime, the number of times these loops are traversed is actually bounded, as the following argument suggests. Suppose that the register is multiplied with $l_1, l_2, \cdots, l_m$ while reading some input string w of length n, resulting in the register value $l = l_1 l_2 \cdots l_m (\#^{-1})^k$, where $k \in \mathbb{N}$, at the end of the computation. If w is accepted by the machine, l should

satisfy the following, as well as being equal to the identity element:

$$l_i = \begin{cases} (\#^{-1})^p x_i \# \text{ for some } p \in \mathbb{N}, & \text{if } x_i \text{ is a negative generator} \\ x_i \#, & \text{if } x_i \text{ is a positive generator} \end{cases}$$

This is called a *permissible padding* in [33]. By looking at the transition function of $\mathcal{E}'$, one can see that the register is multiplied by a # only when an input symbol is consumed. Hence, the number of #'s that occur in l is less than or equal to the length of the string. The register is multiplied with $\#^{-1}$ without consuming any input symbol. In order for the #'s and $\#^{-1}$'s to cancel each other, they should be equal in number. Therefore, it can be concluded that the ε-loops are traversed at most n times.

We can conclude that any context-free language can be recognized by a weakly linear-time bounded free group automaton. Since $\mathbf{F}_2$ contains every free group of countable rank, the proof is complete. □

We state the following theorem, which is the linear-time equivalent of Fact 3.8 [30].

Theorem 3.25. *Suppose G is a finitely generated group and H is a subgroup of finite index. Then $\mathfrak{L}(G)^w_{O(n)} = \mathfrak{L}(H)^w_{O(n)}$.*

Proof. We know that the statement is true in general when there is no time bound by [30]. The proof in [30] still works when all automata in the constructions are required to work in linear time. □

Now we can show that Theorem 3.9 also holds for linear-time bounded group automaton.

Theorem 3.26. $\mathsf{CF} = \mathfrak{L}(\mathbf{F}_2)^w_{O(n)} = \mathfrak{L}(SL(2,\mathbb{Z}))^w_{O(n)} = \mathfrak{L}(GL(2,\mathbb{Z}))^w_{O(n)}$.

Proof. The proof is identical with the proof of Theorem 3.9 by using Theorem 3.25. □

By using the results proven in Subsection 3.3.2, we can demonstrate the language recognition power of weakly linear-time bounded **H**-automata.

Theorem 3.27. $\mathfrak{L}(\boldsymbol{H})^w_{O(n)} \subsetneq \mathfrak{L}(SL(3,\mathbb{Z}))^w_{O(n)}$.

Proof. $\mathfrak{L}(\mathbf{H})^w_{O(n)} \subseteq \mathfrak{L}(SL(3,\mathbb{Z}))^w_{O(n)}$ since **H** is a subgroup of $SL(3,\mathbb{Z})$. Since the Heisenberg group has polynomial growth function [49], there exists a context-free language which cannot be recognized by any **H**-automaton in polynomial time by Theorem 3.23. Since $\mathsf{CF} = \mathfrak{L}(SL(2,\mathbb{Z}))^w_{O(n)}$ by Theorem 3.26, the result follows. □

Theorem 3.28. *(i) For $k \geq 5$, $\mathfrak{L}(\mathbf{H})^w_{O(n)}$ and $\mathfrak{L}(\mathbb{Z}^k)^w_{O(n)}$ are incomparable.*
(ii) $\mathfrak{L}(\mathbf{H})^w_{O(n)}$ and CF are incomparable.

Proof. *i.* In [35], a weakly linear-time bounded **H**-automaton which recognizes the language $\texttt{MULT} = \{x^p y^q z^{pq} | p, q \geq 0\}$ is constructed. The language MULT cannot be recognized by any $\mathbb{Z}^k$-automaton, since any bounded language in $\mathfrak{L}(\mathbb{Q}^+)$ is semilinear by Fact 2.3.

In [50], it is implicitly proven there exists a uniformly n-dissimilar set of size $\Theta(n^k)$ for the language $L_k = \{0^{a_1}10^{a_2}1\ldots0^{a_k}10^{a_1}10^{a_2}1\ldots0^{a_k}1\}$ for all integers k. For $k = 5$, there exists a uniformly n-dissimilar set of size $\Theta(n^5)$ for the language L_5 and $U_{L_5}(n) \geq n^5$. Since $g_{\mathbf{H}}(n)$ is a polynomial of order 4 [49] and $t(n) = O(n)$, $g_{\mathbf{H}}(t(n)) \in o(U_{L_5}(n))$. By Theorem 3.20, we conclude the result.

ii. The language $\texttt{MULT} = \{x^p y^q z^{pq} | p, q \geq 0\}$ is not a context-free language. Since **H** has a polynomial growth function [49], there exists a context-free language which cannot be recognized by any **H**-automaton in polynomial-time by Theorem 3.23. □

Let us note that L_5 can be recognized by a $\mathbb{Z}^5$-automaton in real time. The existence of the languages L_k can be used to prove the linear-time nondeterministic counter hierarchy, with the help of Theorem 3.20.

Theorem 3.29. $\mathfrak{L}(\mathbb{Z}^k)^w_{O(n)} \subsetneq \mathfrak{L}(\mathbb{Z}^{k+1})^w_{O(n)}$ *for* $k \geq 1$.

Proof. The language $L_{k+1} = \{0^{a_1}10^{a_2}1\ldots0^{a_{k+1}}10^{a_1}10^{a_2}1\ldots0^{a_{k+1}}1\}$ can be recognized by a $\mathbb{Z}^{k+1}$-automaton in real time. While scanning the first $k+1$ segments of 0's, the i'th counter is increased for each scanned 0 as 0^{a_i} is read. In the remainder of the computation, the i'th counter is decreased for each scanned 0 when 0^{a_i} is read.

There exists a uniformly n-dissimilar set of size $\Theta(n^{k+1})$ for the language L_{k+1}, so $U_{L_{k+1}}(n) \geq n^{k+1}$. Since $t(n) = O(n)$ and $g_{\mathbb{Z}^k}(n) = n^k$ [45], $g_{\mathbb{Z}^k}(t(n)) \in o(U_{L_5}(n))$. We conclude by Theorem 3.20. □

A celebrated result of the field of computational complexity, the nondeterministic time hierarchy theorem, will enable us to demonstrate that the computational power $\mathbf{F}_2 \times \mathbf{F}_2$-automata is dependent on the time allotted for their execution.

Fact 3.30. [51] *If $g(n)$ is a time-constructible function, and $f(n+1) = o(g(n))$, then there exists a language which cannot be recognized by any nondeterministic Turing machine in time $f(n)$, but can be recognized by a nondeterministic Turing machine in time $g(n)$.*

Assume that any recursively enumerable language can be recognized by some linear-time $\mathbf{F}_2 \times \mathbf{F}_2$-automaton. One can easily build a nondeterministic Turing machine that simulates such a $\mathbf{F}_2 \times \mathbf{F}_2$-automaton with only a polynomial slowdown. But this would mean that any recursively enumerable language can be recognized by some nondeterministic TM in polynomial time, contradicting Fact 3.30, which implies that there exist languages which can only be recognized by nondeterministic Turing machines which run in at least exponential time. We have proven the following theorem.

Theorem 3.31. $\mathfrak{L}(\mathbf{F}_2 \times \mathbf{F}_2)^w_{O(n)} \subsetneq \mathsf{RE}$.

Using the ability of Turing machines to simulate any finite automaton over a computable matrix group, the statement of the above theorem can be extended as follows.

Theorem 3.32. $\mathfrak{L}(G)^w_{O(n)} \subsetneq \mathsf{RE}$ *for any matrix group* G *whose matrix entries are computable numbers.*

Proof. In Theorem 3.18, we have mentioned that Turing machines can simulate any finite automaton over a computable matrix group. By the nondeterministic time hierarchy theorem, it can be shown that there exist some languages which cannot be recognized by any finite automata over matrix groups in linear time. □

Theorem 3.33. $\mathfrak{L}(\mathbf{F}_2)^w_{O(n)} \subsetneq \mathfrak{L}(\mathbf{F}_2 \times \mathbf{F}_2)^w_{O(n)}$.

Proof. It is obvious that an $\mathbf{F}_2$-automaton can be simulated by an $\mathbf{F}_2 \times \mathbf{F}_2$-automaton. $\mathfrak{L}(\mathbf{F}_2)^w_{O(n)} = \mathsf{CF}$ by Theorem 3.26. The inclusion is proper since the non-context-free language $L = \{a^n b^n c^n | n \geq 0\}$ can be recognized by an $\mathbf{F}_2 \times \mathbf{F}_2$-automaton in real time by using the two registers as two counters. □

In the rest of the section, the linear-time counterparts of the relationships in Figure 3.3 will be stated.

Theorem 3.34. *(i)* $\mathfrak{L}(\mathbb{Q}^+)^w_{O(n)} \subsetneq \mathfrak{L}(SL(2,\mathbb{Q}))^w_{O(n)}$.
(ii) $\mathfrak{L}(\mathbb{Z})^w_{O(n)} \subsetneq \mathfrak{L}(BS(1,2))^w_{O(n)} \not\subseteq \mathsf{CF}$.
(iii) $\mathfrak{L}(SL(2,\mathbb{Z}))^w_{O(n)} \subsetneq \mathfrak{L}(SL(3,\mathbb{Z}))^w_{O(n)}$.
(iv) $\mathfrak{L}(\mathbb{Z}^2)^w_{O(n)} \subsetneq \mathfrak{L}(\mathbf{H})^w_{O(n)}$.
(v) CF *and* $\mathfrak{L}(\mathbb{Z}^k)^w_{O(n)}$ *are incomparable for all* $k \geq 2$.
(vi) $\mathfrak{L}(SL(3,\mathbb{Z}))^w_{O(n)} = \mathfrak{L}(GL(3,\mathbb{Z}))^w_{O(n)}$.
(vii) $\mathsf{REG} = \mathfrak{L}(\mathbf{F}_0)^w_{O(n)} \subsetneq \mathfrak{L}(\mathbf{F}_1)^w_{O(n)} = \mathfrak{L}(\mathbb{Z})^w_{O(n)} \subsetneq \mathfrak{L}(\mathbf{F}_2)^w_{O(n)}$.

Proof. (i,ii,iii,iv) Analogous results where no time bound was imposed on the machines were proven in Theorems 3.16, 3.14, 3.11, and 3.13, respectively. The group automata recognizing the witness languages $L = \{a^{2^{2n+1}} | n \geq 0\}$, $\mathtt{UPOW} = \{a^{2^n} | n \geq 0\}$ and $\mathtt{MULT} = \{x^p y^q z^{pq} | p, q \geq 0\}$ operate in weakly linear time in all cases.

(v) The equivalent result for the general case is given in Fact 3.7. The non-context-free language $L' = \{a^n b^n c^n | n \geq 0\}$ can be recognized by a $\mathbb{Z}^2$-automaton in real time.

(vi) The equivalent result for the general case is given in Theorem 3.12. The result follows by Theorem 3.25.

(vii) The equivalent result for the general case is given in Fact 3.5. $\mathbf{F}_0$ is the trivial group, and any regular language can be recognized by a deterministic finite automaton, which can be seen as finite automaton over $\mathbf{F}_0$, in real time. Since $\mathbf{F}_1$ is isomorophic to $\mathbb{Z}$, the equality is obvious. Since the nonregular language $L = \{a^n b^n | n \geq 0\}$ can be recognized by a $\mathbb{Z}$-automaton in real time, the proper inclusion follows. Lastly, since $\mathfrak{L}(\mathbf{F}_2)^w_{O(n)}$ is equivalent to CF by Theorem 3.26, the last proper inclusion is still valid.

□

The results are summarized in Figure 3.4.

3.4. Decision Problems for Matrix Semigroups

So far, we have focused on the langauge recognition power of extended finite automata over matrix groups. In this section, our aim is to make a connection between the theory of extended finite automata and the decision problems for matrix semigroups. Matrices play an important role in various areas of computation, which makes it interesting to study decision problems on matrices. Even for integer matrices of low dimension, many decision problems become non-trivial for finitely generated infinite semigroups.

For our purposes, we define S-automata or extended finite automata over semigroups, generalizing the notion of M-automata from monoids to semigroups. The emptiness problem is defined as the problem of deciding whether a given machine accepts any string. By using the decidability of the emptiness problem of the corre-

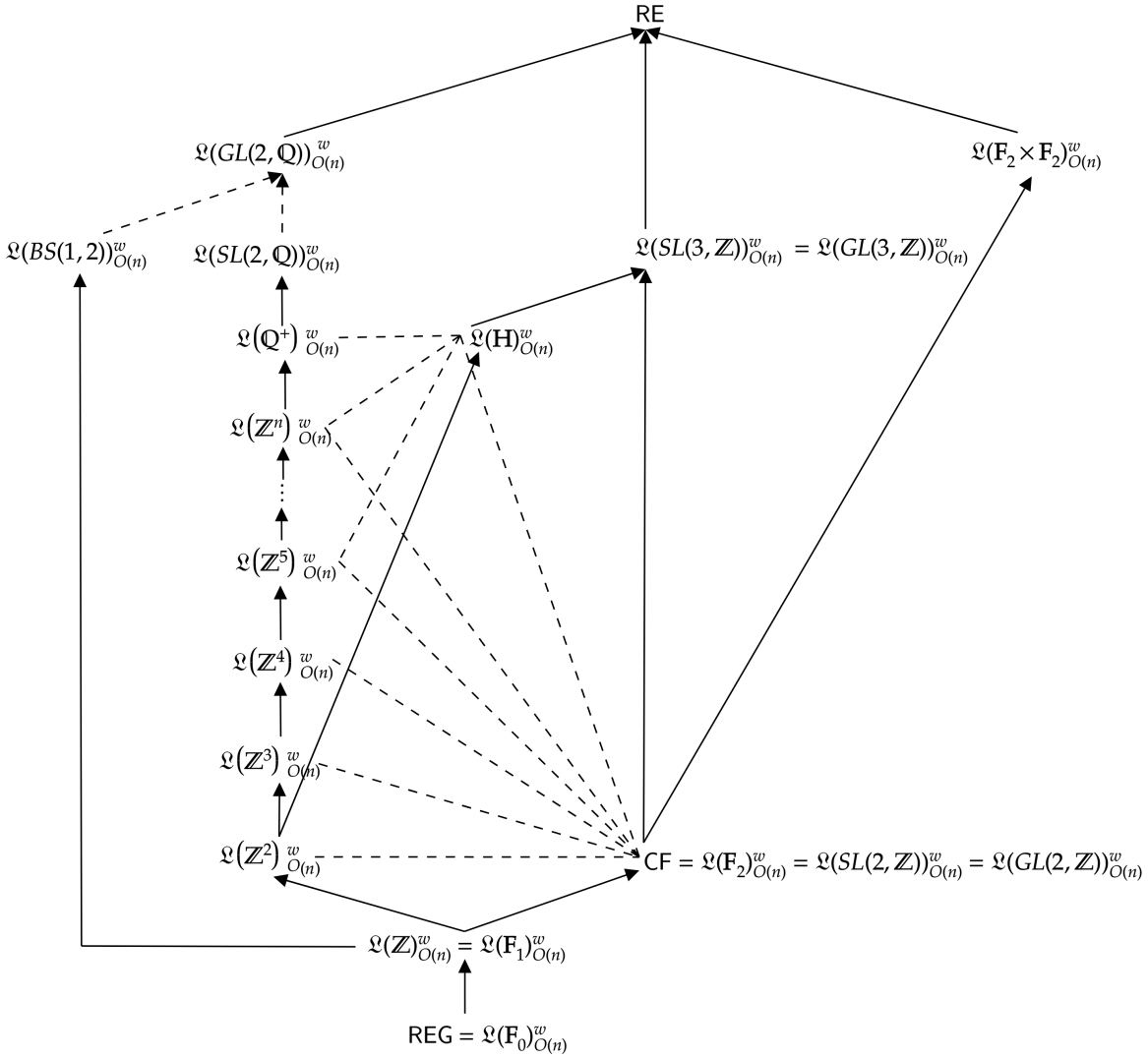


Figure 3.4. Language classes recognized by weakly linear-time bounded group automata

sponding extended finite automata, we provide an alternative proof for the decidability of subsemigroup membership problem for $GL(2,\mathbb{Z})$ and the decidability of the identity problem for $M_2(\mathbb{Z})$. We show that the emptiness and universe problems for extended finite automata over $SL(4,\mathbb{Z})$ are undecidable, using the fact that the subgroup membership problem for $SL(4,\mathbb{Z})$ is undecidable. We also prove some results on the the decidability of the universe problem for extended finite automata, the problem of deciding whether a given machine accepts every string.

3.4.1. Background

Before proceeding to discuss our results, it is necessary to talk about some key definitions and previous studies on the decidability problems for matrix semigroups.

In the following sequel, let G be a finitely generated group.

Decidability is one of the popular topics of combinatorial group theory. In 1911, Dehn proposed several decision problems for groups including the famous word problem [52]. Let us recall the definition for the word problem of a group G.

Word problem for G: Given an element $g \in G$, the problem is to decide whether g represents the identity element.

Explicitly introduced for the first time by Mikhailova [53], a generalization of the word problem which is also known as the *generalized word problem* is the following:

Subgroup membership problem for G: Given $\{g_1, g_2, \ldots, g_n\} \in G$ and an element $g \in G$, the problem is to decide whether g belongs to the subgroup generated by the elements $g_1, g_2, \ldots, g_n$.

In [53], it is proven that subgroup membership problem for $\mathbf{F}_2 \times \mathbf{F}_2$ is undecidable, which yields the undecidability of the subgroup membership problem for $SL(4, \mathbb{Z})$.

One can also consider *submonoid membership problem for group G* and *subsemigroup membership problem for group G*, in which case the problem is to decide whether g belongs to the submonoid and subsemigroup generated by $\{g_1, g_2, \ldots, g_n\}$, respectively.

A further generalization is the rational subset membership problem, as the notion of rational subset generalizes subgroups, submonoids and subsemigroups.

Rational subset membership problem for G: Given a rational subset R of G specified using a finite automaton and $g \in G$, the problem is to decide whether g belongs to R.

Note that the hardness of the problems are in increasing order and the decidability of rational subset problem implies the decidability of the other problems. Similarly, the undecidability of the subgroup membership problem implies the undecidability of the remaining problems.

When we talk about the membership problems for matrices, it is more natural to consider matrix monoids or semigroups. Hence in the above definitions, we may replace the group G with a semigroup S or a monoid M.

For matrices, the well studied decision problem is the subsemigroup membership problem. For 3×3 matrices with integer entries, the subsemigroup membership problem for $M_3(\mathbb{Z})$ is known to be undecidable due to a result by Paterson [54]. The problem remains open for $GL(3,\mathbb{Z})$ and recently it has been proven that it is decidable for a subgroup of $GL(3,\mathbb{Z})$, the Discrete Heisenberg group [55].

An extensive study has been carried out for the matrices from $M_2(\mathbb{Z})$. In [56], it is proven that subsemigroup membership problem for $GL(2,\mathbb{Z})$ is decidable. The result is extended by Potapov and Semukhin in [57] to subsemigroups of matrices from $GL(2,\mathbb{Z})$ extended by singular matrices and in [58] to subsemigroups of nonsingular matrices from $M_2(\mathbb{Z})$. The problem is still open for the general case of $M_2(\mathbb{Z})$.

When the subsemigroup membership problem is asked for the identity matrix, we obtain the identity problem.

Identity problem for S: Given matrices $\{Y_1, Y_2, \ldots, Y_n\} \in S$, the problem is to decide whether the identity matrix I belongs to the semigroup generated by the elements $Y_1, Y_2, \ldots, Y_n$.

Note that the identity problem is a special case of the subsemigroup membership problem. The identity problem is also equivalent to the *group problem*, i.e. the problem of deciding whether a subset of a given semigroup generates a nontrivial group.

The decidability of the identity problem for $M_3(\mathbb{Z})$ is still open. In [59], it was proven that the identity problem for the discrete Heisenberg group $\mathbf{H}$ is decidable (decidability of the membership problem for $\mathbf{H}$ was unknown at the time). The decidability of the identity problem for $SL(4,\mathbb{Z})$, which was open for a long time, was established in [59, 60].

3.4.2. S-automaton

An S-automaton is an extended finite automaton where the group/monoid condition is loosened to a semigroup. In order to define the initialization and acceptance steps, we need an identity element. If S is a monoid or a group, then an identity element already exists and belongs to S. Otherwise, we define 1 to be the identity element of S. Note that when S is not a monoid nor a group, then $\mathcal{E}$ can accept only the empty string. Nevertheless, we define the concept of S-automaton so that the machines in the proofs of Theorem 3.42 and 3.46 are constructed properly.

The *Emptiness problem* for an automaton is the problem of deciding whether the language recognized by the machine is empty. *Universe problem* is the problem of deciding whether the automaton accepts every string.

3.4.3. Decidability of the Subsemigroup Membership Problem for $GL(2,\mathbb{Z})$

It is proven that the subsemigroup membership problem for $GL(2,\mathbb{Z})$ is decidable in [56]. We are going to provide an alternative, automata theoretic proof. We will start by proving a series of lemmas.

For a finite index subgroup H of some finitely generated group G, it is known that $\mathfrak{L}(H) = \mathfrak{L}(G)$ by Fact 3.1. We will go over the proof details and use Fact 3.1 to

show that given a G-automaton, one can construct an H-automaton recognizing the same language.

Lemma 3.35. *Let G be a finitely generated group and let H be a subgroup of finite index. Any G-automaton can be converted into an H-automaton recognizing the same language.*

Proof. Let A be the generator set for G and let $X = A \cup A^{-1}$. Let $\mathcal{E}$ be a G-automaton recognizing language L over alphabet Σ. Then there exists a rational subset $R \subseteq \Sigma^* \times G$ such that $L = \{w \in \Sigma^* | wR1\}$. One can define the elements of G in terms of X to obtain a rational subset $R_0 \subseteq \Sigma^* \times X^*$.

Since H has finite index in G, $W(G) \in \mathfrak{L}(H)$ ([30] Lemma 2.4). It follows that there exists a rational subset $S \subseteq X^* \times H$ such that $W(G) = \{w \in X^* | wS1\}$.

Then the composition $R_0 \circ S$ is a rational subset of $\Sigma^* \times H$ and it follows that $L = \{w \in \Sigma^* | w(R_0 \circ S)1\}$ ([30], Theorem 3.1). The detailed construction of the finite automaton recognizing the composition is given in [28] (Theorem 5.3). Hence a finite automaton $\mathcal{F}$ over $\Sigma^* \times H$ recognizing L exists, from which an H-automaton $\mathcal{E}'$ recognizing L can be constructed. □

The following construction of a pushdown automaton simulating an $\mathbf{F}_2$-automaton is left as an exercise in [37]. We present here some details of the construction.

Lemma 3.36. *Any $\mathbf{F}_2$-automaton can be converted into a pushdown automaton recognizing the same language.*

Proof. Let $\mathcal{E}$ be an $\mathbf{F}_2$-automaton recognizing language L over Σ with the state set Q and let $A = \{a, b\}$ be the generator set for $\mathbf{F}_2$. Let us construct a pushdown automaton $\mathcal{A}$ recognizing the same language with the stack alphabet A. Let $(q', f) \in \delta(q, \sigma)$ be a

transition of $\mathcal{E}$ where $q, q' \in Q$, $\sigma \in \Sigma_\varepsilon$ and $f \in \mathbf{F}_2$ such that

$$f = f_1 f_2 \dots f_n \text{ where } f_i \in X = A \cup A^{-1} \text{ for } i = 1 \dots n.$$

In $\mathcal{A}$, we need additional n states $q_1 \dots q_n \notin Q$ to mimic each given transition of $\mathcal{E}$. If $f_i = a$ or $f_i = b$, then this corresponds to pushing a or b to the stack, respectively. Similarly, if $f_i = a^{-1}$ or $f_i = b^{-1}$, then $\mathcal{A}$ pops a or b from the stack. Each single transition of $\mathcal{E}$ is accomplished by the pushdown automaton $\mathcal{A}$ by going through the additional states and pushing and popping symbols. Initially, the register of $\mathcal{E}$ is initialized with the identity element of $\mathbf{F}_2$, which corresponds to the stack of $\mathcal{A}$ being empty. The acceptance condition of $\mathcal{E}$, which is ending in an accept state with the register being equal to the identity element is realized in $\mathcal{A}$ by starting with an empty stack and accepting with an empty stack in an accept state. We conclude that $\mathcal{A}$ recognizes language L. □

Lemma 3.37. *The emptiness problem for $GL(2,\mathbb{Z})$-automaton is decidable.*

Proof. Let $\mathcal{E}$ be a $GL(2,\mathbb{Z})$-automaton. Since $\mathbf{F}_2$ has finite index in $GL(2,\mathbb{Z})$, one can construct an $\mathbf{F}_2$-automaton recognizing $L(\mathcal{E})$ by Lemma 3.35. The $\mathbf{F}_2$-automaton can be converted to a pushdown automaton $\mathcal{A}$ using the procedure described in Lemma 3.36. Since the emptiness problem for pushdown automata is known to be decidable, we conclude that the emptiness problem for $GL(2,\mathbb{Z})$-automata is also decidable since any $GL(2,\mathbb{Z})$-automaton can be converted to a pushdown automaton. □

Now we prove the main theorem of the section and establish the connection between the emptiness problem for G-automata and the subsemigroup membership problem for G.

Theorem 3.38. *Let G be a finitely generated group. If the emptiness problem for G-automata is decidable, then the subsemigroup membership problem for G is decidable.*

Proof. Let H be a finitely generated subsemigroup of G and let $g \in G$ be given. We are going to construct a G-automaton $\mathcal{E}_1$ and show that $g \in H$ iff $L(\mathcal{E}_1)$ is nonempty. The state transition diagram of $\mathcal{E}_1$ is given in Figure 3.5. The transition labeled by (a, h_i) stands for each one of the transitions that multiply the register with h_i for $i = 1 \dots n$.

$\mathcal{E}_1$ has two states: the initial state q_1 and the accept state q_2. The transition function of $\mathcal{E}_1$ is defined as $\delta(q_1, a) = (q_2, g^{-1})$ and $\delta(q_2, a) = (q_2, h_i)$ for each $i = 1 \dots n$, where the set $\{h_1, \dots, h_n\}$ generates H.

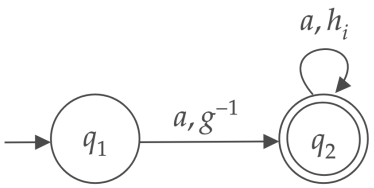


Figure 3.5. State transition diagram of $\mathcal{E}_1$

Note that g^{-1} exists and belongs to G since G is a group. If $g \in H$, then there exists an integer $k \geq 1$ and $i_1, i_2, \dots, i_k \in \{1, \dots, n\}$ such that $h_{i_1} h_{i_2} \cdots h_{i_k} = g$. The string a^{k+1} is accepted by $\mathcal{E}_1$ as the register is initially multiplied by g^{-1} and there exists a product of elements yielding g, from which we can conclude that the identity element can be obtained through a series of transitions of the machine $\mathcal{E}_1$. Hence, we can conclude that $L(\mathcal{E}_1)$ is nonempty.

For the other direction, assume that $L(\mathcal{E}_1)$ is nonempty, which means that some input string is accepted by $\mathcal{E}_1$. Since the acceptance condition requires that the product of the elements multiplied by the register of $\mathcal{E}_1$ is equal to the identity element and the register is initially multiplied by g^{-1}, we can conclude that H contains g.

Now suppose that the emptiness problem for G-automaton is decidable. Then one can check if g is an element of H by constructing $\mathcal{E}_1$ and checking if $L(\mathcal{E}_1)$ is nonempty. Hence, the subsemigroup membership problem G is also decidable. $\square$

Remark: After obtaining the results reported above, we learned that a stronger version of Theorem 3.38 was already proven in [61] and addressed later in [62, 63].

Fact 3.39. [61] *Let G be a finitely generated group, and M a finitely generated monoid. Then the following are equivalent:*

(i) The rational subset problem for $G \times M$ is decidable;
(ii) The membership is decidable for G-automaton subsets of M.

Now we are ready to state our main result.

Theorem 3.40. *The subsemigroup membership problem for $GL(2, \mathbb{Z})$ is decidable.*

Proof. From Lemma 3.37, the emptiness problem for $GL(2, \mathbb{Z})$-automaton is decidable. Then by Theorem 3.38, the result follows. □

Note that we cannot extend this result to $M_2(\mathbb{Z})$. Even though the emptiness problem for $M_2(\mathbb{Z})$-automata is decidable, the construction in Theorem 3.38 works only for group automata.

3.4.4. Decidability of the Identity Problem for $M_2(\mathbb{Z})$

In this section we provide an alternative proof for the decidability of the identity problem for $M_2(\mathbb{Z})$, which was originally proven in [56].

We should first show that the emptiness problem for $M_2(\mathbb{Z})$-automaton is decidable.

Lemma 3.41. *The emptiness problem for $M_2(\mathbb{Z})$-automaton is decidable.*

Proof. Let $\mathcal{E}$ be an $M_2(\mathbb{Z})$-automaton. Any transition labeled by a matrix whose determinant is not equal to ± 1 can be safely removed from $\mathcal{E}$, since after multiplication

with such a matrix, it is not possible that the register is equal to identity matrix again and what we obtain is a $GL(2,\mathbb{Z})$-automaton. The emptiness problem for $GL(2,\mathbb{Z})$-automata is decidable by Lemma 3.37.

□

In the next theorem, we make a connection between the identity problem for a semigroup S and the emptiness problem for the corresponding S-automaton.

Theorem 3.42. *Let S be a finitely generated semigroup. If the emptiness problem for S-automata is decidable, then the identity problem for S is decidable.*

Proof. We are going to construct an S-automaton $\mathcal{E}_2$ and show that S contains the identity element iff $L(\mathcal{E}_2)$ is nonempty. The state transition diagram of $\mathcal{E}_2$ is given in Figure 3.6. The transitions labeled by (a, s_i) stand for each one of the transitions that multiply the register with s_i for $i = 1 \dots n$.

$\mathcal{E}_2$ has two states: the initial state q_1 and the accept state q_2. The transition function of $\mathcal{E}_2$ is defined as $\delta(q_1, a) = (q_2, s_i)$ and $\delta(q_2, a) = (q_2, s_i)$ for each $i = 1 \dots n$ where $\{s_1, s_2, \dots s_n\}$ is the generator set for S.

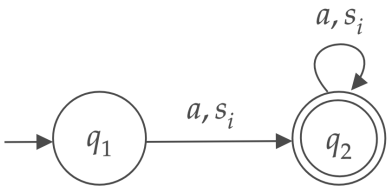


Figure 3.6. State transition diagram of $\mathcal{E}_2$

If S contains the identity element, then there exists an integer $k \geq 1$ and $i_1, i_2, \dots, i_k \in \{1, \dots, n\}$ such that $s_{i_1} s_{i_2} \cdots s_{i_k} = 1$. Then the string a^k is accepted by $\mathcal{E}_2$ as there exists a product of elements yielding the identity element and this product can be obtained by a series of transitions. Hence, we can conclude that $L(\mathcal{E}_2)$ is nonempty. For the converse, suppose that $L(\mathcal{E}_2)$ is nonempty, which means that some

input string is accepted by $\mathcal{E}_2$. Since the acceptance condition requires that the product of the elements multiplied by the register of $\mathcal{E}_2$ is equal to the identity element, we can conclude that S contains the identity element.

Now suppose that the emptiness problem for S-automata is decidable. Then one can check if S contains the identity element by constructing $\mathcal{E}_2$ and checking if $L(\mathcal{E}_2)$ is nonempty. Hence, the identity problem for S is also decidable. □

We connect the two results and state the following.

Theorem 3.43. *The identity problem for $M_2(\mathbb{Z})$ is decidable.*

Proof. By Lemma 3.41, the emptiness problem for $M_2(\mathbb{Z})$-automaton is decidable. The result follows by Theorem 3.42. □

3.4.5. Undecidability of the Decision Problems for $SL(4,\mathbb{Z})$-automata

In this section we are going to prove undecidability results for the emptiness and universe problems of $SL(4,\mathbb{Z})$-automata.

To prove the undecidability of the universe problem for $SL(4,\mathbb{Z})$-automata, we first prove the following theorem.

Theorem 3.44. *Let G be a finitely generated group. If the universe problem for G-automata is decidable, then the subsemigroup membership problem for G is decidable.*

Proof. We are going to construct a G-automaton $\mathcal{E}_3$ such that $g \in H$ iff $L(\mathcal{E}_3) = \Sigma^*$ where $\Sigma = \{a\}$. $\{h_1, h_2, \ldots h_n\}$ is the generator set for H. The state transition diagram of $\mathcal{E}_3$ is given in Figure 3.7. The transition labeled by (a, h_i) and (ε, h_i) stands for each one of the transitions that multiply the register with h_i for $i = 1 \ldots n$.

The rest of the proof is similar to the proof of Theorem 3.38 and omitted here. □

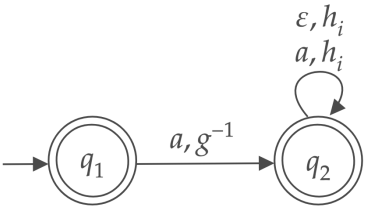


Figure 3.7. State transition diagram of $\mathcal{E}_3$

Now we state the undecidability of the emptiness and universe problems for $SL(4,\mathbb{Z})$-automata.

Theorem 3.45. *Let S be a subsemigroup of $SL(4,\mathbb{Z})$. The emptiness and universe problems for S-automata is undecidable.*

Proof. Since the subgroup membership problem for $SL(4,\mathbb{Z})$ is undecidable [53], by Theorem 3.38 and by Theorem 3.44, the emptiness and universe problems for $SL(4,\mathbb{Z})$-automata are undecidable. □

So far, we have established some connections between decision problems for groups and semigroups and the corresponding automata. Let us also state the following theorem, which links the identity problem for semigroups and the universe problem for the corresponding automata, for the sake of completeness.

Theorem 3.46. *Let S be a finitely generated semigroup. If the universe problem for S-automata is decidable, then the identity problem for S is decidable.*

Proof. We are going to construct an S-automaton $\mathcal{E}_4$ such that S contains the identity element iff $L(\mathcal{E}_4) = \Sigma^*$ where $\Sigma = \{a\}$. $\{s_1, s_2, \ldots s_n\}$ is the generator set for S. The state transition diagram of $\mathcal{E}_4$ is given in Figure 3.8. The transition labeled by (a, s_i) and (ε, s_i) stands for each one of the transitions that multiply the register with s_i for $i = 1 \ldots n$.

The rest of the proof is similar to the proof of Theorem 3.42 and omitted here. □

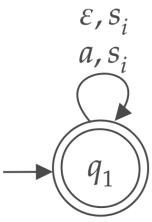


Figure 3.8. State transition diagram of $\mathcal{E}_4$

Note that the converses of Theorem 3.44 and 3.46 are not true. For a given pushdown automaton, an $\mathbf{F}_2$-automaton recognizing the same language can be constructed [37]. It is a well known fact that the universe problem for pushdown automata is undecidable, from which we can conclude that the universe problem for $\mathbf{F}_2$-automaton is undecidable. On the other hand, $\mathbf{F}_2$ is a subgroup of $SL(2,\mathbb{Z})$ and the membership problem for $SL(2,\mathbb{Z})$ and thus the identity problem are known to be decidable [59].

3.5. Open Questions

In this section, we are going to list some questions in need of further investigation.

- Does there exist an $SL(3,\mathbb{Z})$-automaton recognizing $W(\mathbb{Z}^3)$? Corollary 2 of [42] states that the word problem of a finitely generated Abelian group H is recognized by a G-automaton if and only if H has a finite index subgroup isomorphic to a subgroup of G. That corollary could be used to give an affirmative answer to this open question. Unfortunately, the corollary is wrong: Let H be an Abelian group and let $G = \mathbf{F}_2 \times \mathbf{F}_2$. $\mathfrak{L}(\mathbf{F}_2 \times \mathbf{F}_2)$ contains the word problem of any finitely generated Abelian group. Since $\mathbf{F}_2 \times \mathbf{F}_2$ is finitely generated, any finite index subgroup of $\mathbf{F}_2 \times \mathbf{F}_2$ is also finitely generated. Any finite index subgroup of $\mathbf{F}_2 \times \mathbf{F}_2$ is either free or has a subgroup of finite index that is a direct product of free groups [64]. Any subgroup of an Abelian group is again Abelian. Hence, it is not possible that G has a finite index subgroup isomorphic to a subgroup of H.
- Can we describe the necessary properties of a group G so that $\mathfrak{L}(G)$ contains $W(\mathbf{F}_2)$?

- Little is known about $BS(1,2)$-automata. Does $\mathfrak{L}(BS(1,2))$ contain every context-free language?
- Which, if any, of the subset relationships in Figure 3.3 are proper inclusions?
- Can we add other classes above RE in Figure 3.3 by examining groups on matrices with uncomputable entries?
- Theorem 3.20 uses the definition of uniform n-dissimilarity requiring that $g_G(t(n)) \in o(U_L(n))$. Would the theorem be still true if we replace $U_L(n)$ by $A_L(n)$? The gap between $U_L(n)$ and $A_L(n)$ might be large as mentioned in [50]. Consider the language $L = \{a^i b^j | i \neq j\}$. It is stated in [50] that a set of uniformly n-dissimilar strings for L cannot contain more than two strings. However, $A_L(n) \notin O(1)$, since L is not a regular language.
- Can real-time $\mathbf{F}_2$-automata recognize every context-free language?
- Can we prove a stronger version of Theorem 3.23, which is independent of the time component? For instance, for the case of $\mathbf{F}_2$, is it true that $W(\mathbf{F}_2) \notin \mathfrak{L}(\mathbf{H})$ in general?
- The decidability of the membership problem for $GL(3,\mathbb{Z})$ and the identity problem for $M_3(\mathbb{Z})$ are still open. We propose that investigating the decidability of the emptiness and universe problems for extended finite automata defined over 3×3 integer matrices is one possible way for obtaining results about the decision problems on these matrix semigroups.

4. HOMING VECTOR AUTOMATA

The idea of augmenting the classical finite automaton model with an external storage unit that can hold unlimited amounts of information, yet can be accessed in a limited mode, is a celebrated topic of automata theory. In this chapter we introduce homing vector automaton, a finite automaton equipped with a vector which can multiply its vector with an appropriate matrix at each step and can check the entire vector for equivalence to the initial value of the vector.

Matrices are fundamental objects in mathematics and computer science. They are also crucial in automata theory as many finite automaton models such as probabilistic and quantum can be simulated by vector matrix multiplications. Likewise, the vector matrix multiplication view of programming forms the basis of the computation process of homing vector automata.

Homing vector automata are also closely linked to finite automata over matrix groups which we have discussed in Chapter 3. Although in both models the computation is carried out by a series of matrix multiplications, the nature of the registers and the acceptance conditions differentiate the two models.

We examine homing vector automata under several different regimes, enabling us to determine the effect of definitional parameters such as whether the input is scanned in real-time or pausing the head on an input symbol for several steps is allowed, whether the machine can read its register during computation or is blind, with acceptance possible only if the register has returned to its initial value at the end, and whether nondeterminism confers any additional recognition power over deterministic programs.

Another way in which one can examine the nature of the computational power of homing vector automata is by examining models in which the matrices used at each step for transforming the vectors are restricted in some way. Although the definition allows arbitrary rational matrices, one may constrain the matrix entries to belong to

a particular set. In most automaton algorithms in this chapter, the entries of the matrices belong to the set $\{-1, 0, 1\}$, as this basic set will be seen to already capture many capabilities of homing vector automata. Let us note that multiplications with matrices whose entries belong to this set can be used to perform additions, subtractions, resets, and swaps between the vector entries. It is possible to recognize some of the languages in the following discussion with homing vector automata of lower dimension when a larger set of matrix entries is allowed.

The rest of this chapter is structured in the following way:

In Section 4.1, we introduce homing vector automaton, giving the definitions for one-way, real-time, blind and non-blind versions. We begin with some observations about homing vector automata in Section 4.2. A method we use for encoding strings on an alphabet of arbitrary size in a blind homing vector automaton, based on Stern-Brocot tree [65, 66], may be of independent interest and is presented in Section 4.3. In Section 4.4, we investigate the relationship between counter automata and HVAs. In Section 4.5, we establish a connection between the nondeterministic one-way blind version of the HVA model and the extended finite automata, and use this link to prove that these machines can recognize any Turing recognizable language, even when the vector dimension is restricted to four. We then focus on HVAs with real-time access to their input in Section 4.6. We analyze the relationships between different versions of HVAs, present some closure properties, and analyze their stateless versions. Section 4.7 lists some open questions.

4.1. Definitions

Generalizing the idea of finite automaton equipped with a register, we have previously introduced in [9] the vector automaton, a finite automaton which is endowed with a vector, and which can multiply this vector with an appropriate matrix at each step. We give the definition for the real-time deterministic version.

A *real-time k-dimensional deterministic vector automaton* (DVA(k)) [9] is a 7-tuple

$$\mathcal{V} = (Q, \Sigma, \mathrm{M}, \delta, q_1, Q_a, v),$$

where M is a finite set of $k \times k$-dimensional rational-valued matrices, v is the initial (k-dimensional, rational-valued) row vector, and δ is the transition function defined as

$$\delta : Q \times \Sigma_\$ \times \Omega \to Q \times \mathrm{M}.$$

Let $w \in \Sigma^*$ be a given input. The automaton $\mathcal{V}$ reads the sequence $w\$$ from left to right symbol by symbol. It uses its states and its vector to store and process the information. In each step, it can check whether the first entry of the vector is equal ($=$) to 1 or not ($\neq$). We call this feature the "status" of the first entry and represent it by the set $\Omega = \{=, \neq\}$.

The details of the transition function are as follows. When $\mathcal{V}$ is in state $q \in Q$, reads symbol $\sigma \in \Sigma_\$$, and the first entry status is $\omega \in \Omega$, the transition $\delta(q, \sigma, \omega) = (q', A)$ results in $\mathcal{V}$ entering state $q' \in Q$, and its vector being multiplied by $A \in \mathrm{M}$ from the right.

At the beginning of the computation, $\mathcal{V}$ is in state q_1 and the vector is v. The initial vector is freely chosen by the designer of the automaton. Then, after reading each symbol, the state and vector are updated according to the transition function as described above. Thus the vector $v^{(i)}$ at step i is obtained by multiplying the vector $v^{(i-1)}$ at step $i-1$ by a specified matrix A so that $v^{(i)} = v^{(i-1)}A$. The input w is accepted if the final state is an accept state and the first entry of the final vector is 1 after processing the right end-marker \$. Otherwise, the input is rejected. The set of all accepted strings is said to be the language recognized by $\mathcal{V}$.

As the acceptance condition for many of the classical models requires that the register is equal to its initial value at the end of the computation, we adopt the same requirement for vector automata and propose homing vector automata.

A *k-dimensional homing vector automaton* (HVA(k)) is a 7-tuple

$$\mathcal{V} = (Q, \Sigma, \mathrm{M}, \delta, q_1, Q_a, v),$$

where M is a set of $k \times k$ rational valued matrices and v is an initial row vector with rational entries, as in the definition of vector automaton.

A HVA is different from a vector automaton in two ways: (1) Homing vector automata do not read the right end-marker after reading the input, so there is no chance of postprocessing and, (2) instead of checking the status of the first entry, a homing vector automaton checks whether the complete current vector is identical to the initial vector or not.

Formally, the transition function of a *one-way k-dimensional deterministic homing vector automaton* (1DHVA(k)) is defined as

$$\delta : Q \times \Sigma \times \Omega \ \rightarrow Q \times \mathcal{D} \times \mathrm{M},$$

such that Ω is the set $\{=, \neq\}$, where $=$ indicates equality to the initial vector v, and $\neq$ otherwise, $\mathcal{D}$ is the set of head directions $\{\downarrow, \rightarrow\}$ and M is a set of $k \times k$ rational-valued matrices. The initial vector is freely chosen by the designer of the automaton.

Specifically, $\delta(q, \sigma, \omega) = (q', d, A)$ means that when $\mathcal{V}$ consumes $\sigma \in \Sigma$ in state $q \in Q$, with its current vector corresponding to $\omega \in \Omega$ (ω having the value $=$ if and only if the current vector equals the initial vector), it switches to state $q' \in Q$, multiplying its current vector with the matrix $A \in \mathrm{M}$ on the right and moving the tape-head in direction $d \in \mathcal{D}$.

A *one-way k-dimensional deterministic blind homing vector automaton* (1DBHVA(k)) is a restricted 1DHVA(k) which is not allowed to check the vector until the end of the computation. The transition function δ is defined as

$$\delta : Q \times \Sigma \to Q \times \mathcal{D} \times \mathrm{M},$$

so that the next move of the machine does not depend on the current status of the vector.

By omitting the tape-head directions and assuming that the tape-head moves right at each step, we obtain the *real-time k-dimensional deterministic homing vector automaton* (DHVA(k)) and *real-time k-dimensional deterministic blind homing vector automaton* (DBHVA(k)) models. The ranges of the corresponding transition functions are replaced with $Q \times \mathrm{M}$.

Now we are going to define the nondeterministic versions of homing vector automata. Formally, the transition function of a *one-way k-dimensional nondeterministic homing vector automaton* (1NHVA(k)) is defined as

$$\delta : Q \times \Sigma_\varepsilon \times \Omega \to \mathcal{P}(Q \times \mathrm{M}).$$

The blind version, *one-way k-dimensional nondeterministic blind homing vector automaton* (1NBHVA(k)) is a 1NHVA(k) which is not allowed to check the vector until the end of the computation. The transition function of a 1NBHVA(k) is defined as

$$\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q \times \mathrm{M}).$$

By not allowing ε-moves, we obtain the real-time versions, *real-time k-dimensional nondeterministic homing vector automaton* (NHVA(k)) and *real-time k-dimensional nondeterministic blind homing vector automaton* (NBHVA(k)). The domains of the

transition functions are replaced with $Q \times \Sigma \times \Omega$ and $Q \times \Sigma$ respectively, by replacing Σ_ε with Σ.

An input string w of length n is accepted by a homing vector automaton if there exists a computation in which the machine enters an accept state with the tape-head on the $n+1$'st tape square and the vector is equal to the initial value v.

The abbreviations used for homing vector automata variants discussed so far are given in Table 4.1.

Table 4.1. The abbreviations for HVA variants.

	Real-time	One-way
Deterministic	DHVA(k)	1DHVA(k)
Deterministic blind	DBHVA(k)	1DBHVA(k)
Nondeterministic	NHVA(k)	1NHVA(k)
Nondeterministic blind	NBHVA(k)	1NBHVA(k)

Given a homing vector automaton $\mathcal{V} = (Q, \Sigma, \mathrm{M}, \delta, q_1, Q_a, v)$, we abbreviate it by $\mathrm{HVA}(k)_{\mathrm{M}}$ when we want to specify the set of matrices M used by $\mathcal{V}$. When we want to specify the number of states of a machine, we add an n- (or (n)- to avoid any confusion) to the front of the model name, where $n = |Q|$ is the number of the states.

We will denote the set of $k \times k$ matrices whose entries belong to the set $\{-m, -m+1, \ldots, 0, \ldots, m-1, m\}$ for some positive integer m by $S_k(m)$.

4.2. Some Observations

Homing vector automata are not allowed to perform postprocessing by definition, since the computation ends once they reach the right end-marker. We start by observing that allowing postprocessing does not bring any additional power to NBHVAs and 1NBHVAs.

HVAs using end-marker will be denoted by the abbreviation $\text{HVA}_\$$.

Lemma 4.1. *Let L be a language recognized by a* $\text{XBHVA}_\$(k)$ $\mathcal{V}$*, where* $X \in \{\text{N},\text{1N}\}$*. Then, L is also recognized by a* $\text{XBHVA}(k)$ $\mathcal{V}'$*.*

Proof. First of all, we assume that $\mathcal{V}$ does not contain any \$-transitions that do not lead to an accept state, since any such transitions may be removed from $\mathcal{V}$ without changing the language. We are going to analyze the cases of real-time and one-way computation and in each case, we will start constructing $\mathcal{V}'$ such that $\mathcal{V}'$ mimics the transitions of $\mathcal{V}$ on every possible symbol $\sigma \in \Sigma_\varepsilon$.

If the computation is real-time, then it ends as soon as the right end-marker is processed. We create new transitions to handle the postprocessing, which emulate $\mathcal{V}$'s last action before reading the end-marker (which would end up in an accept state) and the end-marker ($\sigma\$$) at once: At any point during the scanning, if reading σ would cause $\mathcal{V}$ to switch to a state from which the end-marker \$ would lead to an accept state, a new nondeterministic transition takes $\mathcal{V}'$ to the additional state, which is an accept state. During this transition, the register is updated so that the update accounts for both reading σ and \$. All other states of $\mathcal{V}'$, which are inherited from $\mathcal{V}$, are designated to be non-accept states. Thus, $\mathcal{V}'$ simulates the computation of $\mathcal{V}$ on any non-empty string, and accepts the input in some computational path if and only if $\mathcal{V}$ accepts it.

Now suppose that the computation is one-way. Let $Q_\$$ be the set of states of $\mathcal{V}$ that have an outgoing \$-transition. After finishing reading the string, $\mathcal{V}$ should enter a state from $Q_\$$, read the \$ symbol and possibly make some ε-transitions and eventually end in an accept state, to accept any string. Let $G_\$$ be the graph obtained from the transition diagram of $\mathcal{V}$, by removing all transitions except the \$-transitions and ε-transitions. Let r_q be the subgraph of $G_\$$, induced by the set of reachable vertices from q in $G_\$$, for each $q \in Q_\$$. We create a copy of each subgraph r_q and denote it by r_q^c, replace the \$ symbols in r_q^c with ε and connect it to $\mathcal{V}'$: For each incoming transition to q in $\mathcal{V}$, we create a copy of the transition and connect it to the copy of q in r_q^c. The \$-transitions inherited from $\mathcal{V}$ are removed from $\mathcal{V}'$ and any accept state of $\mathcal{V}$ is

no longer an accept state in $\mathcal{V}'$. $\mathcal{V}'$ simulates the computation of $\mathcal{V}$ on any non-empty string until scanning the \$ and then follows the transitions in the newly added states to reach an accept state.

If L contains the empty string, we add one more state that has the following properties: (i) it becomes the new initial state of the resulting machine, (ii) it is an accept state, (iii) it causes the machine to behave (i.e. transition) like the original initial state of $\mathcal{V}$ upon reading the first symbol, and (iv) there is no transition coming in to this state. □

The idea given in the proof of Lemma 4.1 does not apply for non-blind models since the status of the vector may be changed after reading the last symbol of the input (just before reading the right end-marker). In fact, one can show that DHVAs using end-marker are more powerful than ordinary DHVAs in terms of language recognition by the witness language $\texttt{NEQ} = \{a^i b^j | i \neq j\}$.

Theorem 4.2. $\bigcup_k \mathfrak{L}(\text{DHVA}(k)) \subsetneq \bigcup_k \mathfrak{L}(\text{DHVA}(k)_{\$})$.

Proof. The subset inclusion is immediate, since the postprocessing may very well involve multiplication with the identity matrix. For the inequality, consider the language $\texttt{NEQ} = \{a^i b^j | i \neq j\}$. Suppose for a contradiction that there exists a DHVA(k) $\mathcal{V}$ recognizing NEQ for some k. Let v be the initial vector of $\mathcal{V}$. There exist sufficiently long strings $w_1 = a^m b^n$ and $w_2 = a^m b^o$, $m \neq n$, $m \neq o$, $n < o$ such that $\mathcal{V}$ is in the same accept state after reading w_1 and w_2 and the vector is equal to v, since the strings belong to NEQ. When both strings are extended with b^{m-n}, $a^m b^n b^{m-n} \notin \texttt{NEQ}$ whereas $a^m b^o b^{m-n} \in \texttt{NEQ}$. Since the same vector is being multiplied with the same matrices associated with the same states during the processing of the string b^{m-n}, it is not possible for $\mathcal{V}$ to give different responses.

Now let us prove that the language NEQ can be recognized by a DHVA$_{\$}$(2) $\mathcal{V}'$. The initial vector of $\mathcal{V}'$ is $v' = (1 \;\; 1)$, and the state diagram of $\mathcal{V}'$ is given in Figure 4.1.

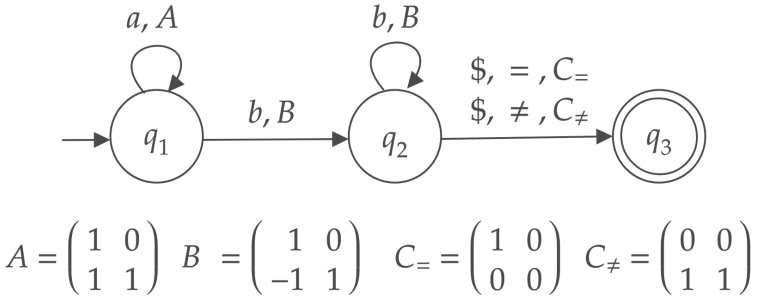


Figure 4.1. State diagram of $\mathcal{V}'$ recognizing NEQ

For each a, the first entry is increased by 1 and for each b the first entry is decreased by 1 using the second entry of the vector, which is equal to 1 throughout the computation. The increment and decrement operations are performed by the matrices A and B.

When reading the end-marker, if the value of the vector is equal to its initial value, meaning that the number of a's and b's were equal to each other, the vector is multiplied with $C_=$, which sets the second entry to 0, so that the input string is not accepted. Otherwise, if the vector is not equal to its initial value, meaning that the number of a's and b's were not equal, the vector is multiplied with $C_{\neq}$, which sets the first entry to 1. This returns the vector to its initial value, and the input string is accepted. □

Any BHVA with end-marker whose matrices are rational valued can be simulated by a BHVA with end-marker and integer valued matrices in the cost of increasing the size of the vector by 2. The proof is due to Abuzer Yakaryılmaz and can be found in [17].

Lemma 4.3. *For any given rational-valued* (n)-XBHVA$_\$(k)$ $\mathcal{V}$*, where* $X \in \{$D,1D,N,1N$\}$*, there exists an integer-valued* (n)-XBHVA$_\$(k+2)$ $\mathcal{V}'$ *that recognizes the same language.*

For nondeterministic HVAs, we can state the following corollary.

Corollary 4.4. *Rational-valued* XBHVA*s and integer-valued* XBHVA*s where* $X \in \{$N,1N$\}$ *recognize the same class of languages.*

Proof. By using Lemma 4.3, we can conclude that any rational-valued XBHVA can be simulated by an integer-valued XBHVA using the end-marker. Then, by using Lemma 4.1, we can remove the end-marker. □

Note that although they recognize the same classes of languages, a rational valued HVA can be simulated by an integer valued HVA in the cost of increasing the vector size by 2 and using some additional states.

4.3. Encoding Strings with Homing Vector Automata

While recognizing certain languages, it may be necessary to hold information about the string that is read so far in the entries of the vector. We call this notion "encoding the string".

In this section, we are going to discuss some encoding techniques that can be performed by homing vector automata. The methods are applicable by the most restricted, real-time deterministic and blind version and therefore can be carried out by any homing vector automata. In the first part, we present the generalized Stern-Brocot encoding which can be performed by k-dimensional homing vector automata using only matrices belonging to the set $S_k(1)$, for any string belonging to a k-letter alphabet. In the second part, we present another encoding method which can be performed by 2-dimensional HVAs, regardless of the size of the alphabet. The method also can be used for base conversion, which may be necessary while recognizing some specific languages.

4.3.1. Stern-Brocot Encoding

The Stern-Brocot tree is an infinite complete binary tree whose nodes correspond one-to-one to positive rational numbers [65, 66]. Crucially for our purposes, the Stern-Brocot tree provides a basis for representing strings as vectors of integers, as suggested for binary alphabets in [67]. The fractions in the Stern-Brocot tree can be stored as vectors of dimension 2, where the vector entries are the denominator and the numerator

of the fraction. This representation allows us to perform the binary encoding easily in homing vector automata, as follows.

The empty string is represented by $(1 \;\; 1)$. Now suppose that we want to encode a binary string w of length n. For $i = 1$ to n, if $w[i] = 0$, we add the value of the first entry to the second one, and if $w[i] = 1$, we add the value of the second entry to the first one, multiplying the vector with the appropriate one of the following matrices M_0 and M_1:

$$M_0 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \qquad M_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

A list of some binary strings and their encodings is as follows. A proof on the uniqueness of the encoding can be found in [67].

0	$(1 \;\; 2)$	00	$(1 \;\; 3)$	10	$(2 \;\; 3)$	000	$(1 \;\; 4)$	010	$(3 \;\; 5)$
1	$(2 \;\; 1)$	01	$(3 \;\; 2)$	11	$(3 \;\; 1)$	001	$(4 \;\; 3)$	011	$(5 \;\; 2)$

Given the vector representation v_w of a string w, it is also possible to decode the string with the following procedure: Let $|w| = n$ and $v_w = (a \;\; b)$. Set $w[n] = 0$ if $b > a$, and $w[n] = 1$ otherwise. Subtract the smaller entry from the larger one to obtain v_w^{n-1} and repeat this routine until you obtain the vector $(1 \;\; 1)$. When the given vector is not a valid representation of a string, then it is not possible to obtain $(1 \;\; 1)$. The matrices required for this procedure are N_0, which has the effect of subtracting the value of the first entry of the vector from the second entry, and N_1, for the symmetric action. Note that $N_0 = (M_0)^{-1}$ and $N_1 = (M_1)^{-1}$.

$$N_0 = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \qquad N_1 = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$$

We generalize the scheme mentioned above to strings on alphabets of arbitrary size and present a new method for encoding strings. Let $\Sigma = \{a_1, a_2, \ldots, a_k\}$, and $w \in \Sigma^*$. With the *generalized Stern-Brocot encoding* method described below, it is possible to uniquely encode w using a vector of size k and $k \times k$ matrices whose entries belong to the set $\{-1, 0, 1\}$.

We start with the k dimensional vector $(1 \;\; 1 \;\; \cdots \;\; 1)$, which represents the empty string. Suppose that $|w| = n$. To encode w, for $i = 1$ to n, if $w_i = a_j$, the vector is multiplied with the matrix E_j^k, the k dimensional identity matrix whose j'th column has been replaced with a column of 1's. Multiplication with E_j^k causes the j'th entry of the vector to be replaced by the sum of all the entries in the vector.

Among the different generalizations of the Stern-Brocot fractions, one that appears in [68] under the name of "Stern's triatomic sequence" is similar to the encoding we propose for the case $k = 3$. The similarity lies in the construction of the sequence, but that sequence is not used for the purpose of encoding. As far as we know, no such generalization exists for the case $k > 3$.

In the following lemma, we prove the uniqueness of this generalized encoding.

Lemma 4.5. *No two distinct strings on Σ $(|\Sigma| = k)$ can be represented by the same vector of size k using the generalized Stern-Brocot encoding.*

Proof. We will prove by induction on n that if a k-dimensional vector v is the generalized Stern-Brocot encoding of a string of length n, then v is not the encoding of any other string of length at most n.

The empty string is represented by the k-dimensional vector of 1's. The claim clearly holds for $n = 0$, since no other strings of at most this length exist. Now assume that the claim holds for all natural numbers up to $n-1$. Let w be a string of length n. The vector v_w representing w is obtained by multiplying the vector v_w^{n-1}, representing the first $n-1$ symbols of w, with E_j^k if $w[n] = a_j$. We will examine various possibilities

regarding this final multiplication. Note that at a single step, it is possible to modify only a single entry of each vector. Now consider any string $u \neq w$ with $|u| = l$ and $l \leq n$. If w and u have the same first $n-1$ symbols, then $v_w^{n-1} = v_u^{l-1}$, the last symbols of the two strings are unequal, and it is not possible to obtain $v_w = v_u$ since the same vector is multiplied by different matrices. In the remaining case, we know by the induction hypothesis that $v_w^{n-1} \neq v_u^{l-1}$. If these vectors disagree in more than two entries, there is no way that one can obtain the same vector by multiplying them once with some matrices of the form E_j^k. So we consider the case of the two vectors disagreeing in at most two entries.

Suppose that v_w^{n-1} and v_u^{l-1} differ only in the i'th entry. If the final multiplications both work on the i'th entries, they will be adding the same number to them, resulting again in vectors differing in their i'th entries. If one or more of the final multiplications deals with another entry, then the final vectors will surely disagree in that entry. It is not possible in any case to end up with equal vectors,

Now suppose that v_w^{n-1} and v_u^{l-1} differ in two entries. If the final multiplications work on the same entry, then the final vectors will disagree in at least one entry. In the only remaining case, each one of the vectors is multiplied by a matrix updating a different one of the disagreeing entries. Let us represent the disagreeing entries of the vectors v_w^{n-1} and v_u^{n-1} by the pairs $(a \;\; b)$ and $(c \;\; d)$, respectively. Let x be the sum of the remaining $k-2$ entries in which the vectors agree. Without loss of generality, say that the entries become $(a \quad a+b+x)$ and $(c+d+x \quad d)$ after the final multiplication. But if the final vectors are equal, these pairs should also be equal, implying $c+b+2x = 0$, an impossibility.

We therefore conclude that it is not possible to have $v_w = v_u$ for any string u of length at most n. □

Like in the binary case, given the vector representation of a string, it is possible to reconstruct the string. The all-ones vector corresponds to the empty string. Any

other vector v_w encoding a string w of length n in this encoding has a unique maximum entry, say at position j. Then $w[n]$ is a_j, and we obtain v_w^{n-1} by subtracting the sum of the other entries from the greatest entry. One repeats this procedure, reconstructing the string from right to left, until one ends up with the all-ones vector. In terms of matrices, multiplications with the inverses of E_j^ks capture this process.

We demonstrate the use of generalized Stern-Brocot encoding in the following example.

Example 4.6. $\texttt{MPAL}_1 = \{w\#w^r | w \in \{a_1, a_2, \ldots, a_l\}^*\} \in \mathfrak{L}(\text{DHVA}(l)_{S_l(1)})$.

Let us construct a $\text{DHVA}(l)_{S_l(1)}$ $\mathcal{V}$ recognizing $\texttt{MPAL}_1$. The input alphabet is $\{a_1, a_2, \ldots, a_l\}$, and the corresponding matrices are $\{E_1^l, E_2^l, \ldots, E_l^l\}$. Starting with the l dimensional vector of 1's, $\mathcal{V}$ encodes the string by multiplying its vector with the matrix E_j^l whenever it reads an a_j until it encounters a # . After reading the #, $\mathcal{V}$ starts decoding by multiplying the vector with matrix $(E_j^l)^{-1}$ whenever it reads an a_j.

If the string is of the form $w\#w^r$, the vector will be multiplied with the inverse matrices in the correct order and the resulting value of the vector will be $(1 \;\; 1 \;\; \cdots \;\; 1)$.

We also need to show that the input string is not accepted when it is not of the form $w\#w^r$. Consider an input string $x\#y^r$ and suppose that it is accepted by $\mathcal{V}$. Let v' denote the vector after reading $x\#$ and let Y denote the product of the matrices the vector is multiplied while reading y^r. Since the string is accepted, $v'Y = (1 \;\; 1 \;\; \ldots \;\; 1)$ must be true. Since the matrices $(E_j^l)^{-1}$ are invertible, Y is also invertible, which implies that v' must be unique. Since $y\#y^r \in \texttt{MPAL}$, then v' must be the vector obtained after reading y . From Lemma 4.5, we know that every string has a unique representation and we conclude that x and y are identical.

4.3.2. Base-m Encoding

In this subsection, we discuss a well-known encoding technique, which can be easily adopted to homing vector automata.

For any $w \in \{1, 2, \ldots, m-1\}^+$ and $m \leq 10$, let $e_m(w)$ be the base-10 number encoded by w in base-m:

$$e_m(w) = m^{|w|-1}w[1] + m^{|w|-2}w[2] + \cdots + m^1 w[|w|-1] + m^0 w[|w|].$$

The encoding $e_m(w)$ can be easily obtained by using vector-matrix multiplications. Starting with the initial vector $v = (1 \;\; 0)$, and multiplying the vector with

$$A_i^m = \begin{pmatrix} 1 & i \\ 0 & m \end{pmatrix}$$

for each symbol i, $e_m(w)$ is obtained in the first entry of the vector.

When the vector is multiplied by A_i^m, the second entry is multiplied by m and then incremented by i. As a result, this process ends up with $e_m(w)$ appearing in the second entry of the vector.

Given $w \in \{1, 2, \ldots, m-1\}^+$, it is also possible to obtain the encoding for w^r, that is $e_m(w^r)$ in the second entry of the vector. This can be accomplished starting with the initial vector $(1 \;\; 0)$ and multiplying the vector with the matrices B_i^m for each symbol in Σ.

$$B_i^m = \begin{pmatrix} m & i \\ 0 & 1 \end{pmatrix}$$

Multiplication by B_i^m increments the second entry by i times the first entry and multiplies the second entry by m. After reading k symbols, the first entry holds m^k.

By the base-m encoding, any number in base-m that does not contain a 0 digit can be converted to its base-10 equivalent. One should be careful if 0 is included in the alphabet, since appending 0s at the beginning of the string wouldn't change its encoding and the encoding wouldn't be unique in such a case. Nevertheless, by letting the alphabet to be $\{0, 1, \ldots, m-1\}$ and using the same matrices as above, any number in base-m can be converted to its base-10 equivalent.

For instance, for the case where the alphabet has size 2, the matrices used for the conversion have the following form:

$$A_0^2 = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \qquad A_1^2 = \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}.$$

Hence given $w \in 1\{0,1\}^*$, one obtains the base-10 integer corresponding to the binary number represented by w in the second entry of the vector upon multiplication by the matrices A_0^2 and A_1^2.

When $m = 10$ and w is a string containing at most 9 different symbols, then note that one can obtain w in the first entry of the vector, by multiplying the vector with A_i^{10} for each symbol $i \in \Sigma = \{0, 1, \ldots, 9\}$.

$$A_i^{10} = \begin{pmatrix} 1 & i \\ 0 & 10 \end{pmatrix}$$

4.4. Relationship with Counter Automata

In this section, we are going to analyze the relationship between counter automata and homing vector automata. In the first part, we will examine the blind case and show that homing vector automata outperform counter automata. In the second part, we will present some incomparability results between real-time non-blind counter automata and homing vector automata.

4.4.1. Blind Counter Automata

We are going to start this section by showing that BHVA(1)'s and kBCA's are equivalent in power.

Theorem 4.7. $\bigcup_k \mathfrak{L}(\text{X}k\text{BCA}) = \mathfrak{L}(\text{XBHVA(1)})$ *where* $X \in \{\text{D,N,1D,1N}\}$.

Proof. Let $\mathcal{C}$ be an XkBCA. We are going to construct a XBHVA(1) $\mathcal{V}$ simulating $\mathcal{C}$. The register of $\mathcal{V}$ is initialized to 1. We are going to choose k distinct primes $\{p_1, \ldots, p_k\}$ to be multiplied with the register of $\mathcal{V}$, each representing a counter. An increment and decrement of the i'th counter of $\mathcal{C}$ is simulated by multiplying $\mathcal{V}$'s register by p_i and $\frac{1}{p_i}$ respectively. A string is accepted by $\mathcal{V}$ if the register is equal to 1 at the end of the computation, that is when all the counters are equal to 0.

Now suppose that we are given a XBHVA(1) $\mathcal{V}$. We are going to construct a XkBCA $\mathcal{C}$ simulating $\mathcal{V}$. We may assume that the register of $\mathcal{V}$ is not multiplied by 0, since such a computation will never be accepting. Let $A = \{a_1, a_2, \ldots, a_n\}$ be the set of all rational numbers the register of $\mathcal{V}$ can be multiplied with. Let $P = \{p_1, p_2, \ldots, p_k\}$ be the set of prime factors of the denominators and the numerators of the rational numbers in A. Then each $a_i \in A$ can be expressed as

$$a_i = (-1)^{t_i} \frac{p_1^{x_{1_i}} p_2^{x_{2_i}} \cdots p_k^{x_{k_i}}}{p_1^{y_{1_i}} p_2^{y_{2_i}} \cdots p_k^{y_{k_i}}},$$

where $t_i = 0$ if a_i is positive and $t_i = 1$ if a_i is negative. $\mathcal{C}$ will have k counters and the state set of $\mathcal{C}$ will consist of two copies of $\mathcal{V}$'s states, $Q \times 0$ and $Q \times 1$. The two copies are needed to encode the sign information of $\mathcal{V}'s$ register in the states of $\mathcal{C}$. Any computation starts in the first copy, in the state $(q_1, 0)$. Suppose that $\mathcal{V}$ moves to state q' from state q by multiplying its register with a_i. This transition is implemented in $\mathcal{C}$ by adding the following transitions: If $t_i = 0$, then the transitions from $(q, 0)$ to $(q', 0)$ and from $(q, 1)$ to $(q', 1)$ exist in $\mathcal{C}$. If $t_i = 1$, then the transitions from $(q, 0)$ to $(q', 1)$ and from $(q, 1)$ to $(q', 0)$ exist in $\mathcal{C}$. In these transitions, the j'th counter is incremented by $x_{j_i} - y_{j_1}$. The only accept states of $\mathcal{C}$ are those of the form (q, t_0) where q is an accept state of $\mathcal{V}$. A string is accepted by $\mathcal{C}$ when all the counters are equal to 0 , that is when the register of $\mathcal{V}$ is equal to 1. □

We list the following equivalences among the models.

- $\mathfrak{L}(\text{1NBHVA(1)}) = \bigcup_k \mathfrak{L}(\text{1N}k\text{BCA}) = \bigcup_k \mathfrak{L}(\mathbb{Z}^k) = \mathfrak{L}(\mathbb{Q}^+) = \mathfrak{L}(\text{1NFAMW})$

In the next theorem, we show that HVA(2)s are more powerful than counter automata. Before proving our result, we first prove a lemma to compare the computational power of real-time and one-way deterministic blind counter automata.

Lemma 4.8. $\bigcup_k \mathfrak{L}(\text{D}k\text{BCA}) = \bigcup_k \mathfrak{L}(\text{1D}k\text{BCA})$.

Proof. We are going to show that any computation by a 1DkBCA can be carried out in real-time by a DkBCA. For each input symbol, there can be only one outgoing transition from each state, since the computation is deterministic. There cannot be any loop in the state diagram which does not move the tape head, since in such a case, the next symbol will never be scanned. The only transitions which don't move the tape head should be in the form schematized in Figure 4.2. We omit the counter updates in the figure.

There should be a sequence of transitions which do not move the tape head when scanning the symbol σ, followed by a transition which moves tape head right, upon

Figure 4.2. A part of the transition diagram of a one-way deterministic blind counter automaton

scanning the same symbol. Any other outcoming arrows from these states cannot be traversed as it is not possible to read any symbol different from σ without moving the tape head. This sequence of transitions can be handled by a single transition which moves the machine from q_{i_1} to q_{i_n}, moves the tape head right and makes the necessary modifications on the counters. Hence a DkBCA recognizing the same language as the original one which operates in real-time can be obtained. □

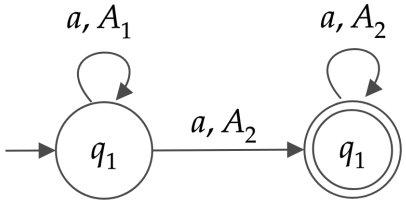
Theorem 4.9. $\bigcup_k \mathfrak{L}(\text{X}k\text{BCA}) \subsetneq \mathfrak{L}(\text{XBHVA}(2))$ *where* $X \in \{\text{D,1D,N,1N}\}$.

Proof. The inclusions follow by Theorem 4.7, as HVA(1)s are capable of simulating counter machines. For the deterministic case, it is known that no DkCA can recognize the language $\texttt{MPAL}_2$ in real-time for any k [22]. Hence we get that no DkBCA and therefore no 1DkBCA (as the two models are equivalent by Lemma 4.8) can recognize $\texttt{MPAL}_2$. On the other hand, $\texttt{MPAL}_2$ can be recognized by a DBHVA(2) as shown in Example 4.6.

For the nondeterministic case, any unary language recognized by a 1NkBCA is regular by Fact 2.2, since $\bigcup_k \mathfrak{L}(1\text{N}k\text{BCA}) = \mathfrak{L}(1\text{NFAMW})$. It is possible to recognize the unary nonregular language $\texttt{UPOW}' = \{a^{n+2^n} | n \geq 1\}$ by a NBHVA(2) $\mathcal{V}$ with the initial vector $v = (1 \;\; 1)$, whose state transition diagram is given in Figure 4.3.

□

Now we show that the same result can be achieved by 3-dimensional nondeterministic HVAs whose matrices are restricted to have integer entries.



$$A_1 = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \quad A_2 = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$$

Figure 4.3. State transition diagram of $\mathcal{V}$ recognizing UPOW′

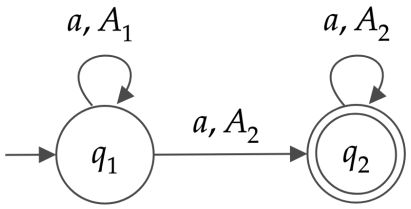
Theorem 4.10. $\bigcup_k \mathfrak{L}(\text{X}k\text{BCA}) \subsetneq \mathfrak{L}(\text{XBHVA}(3)_{M_3(\mathbb{Z})})$ *where* X ∈ {N,1N}.

Proof. Any XkBCA can be simulated by a XBHVA(1) by Theorem 4.7 and any XBHVA(1) can be simulated by a XBHVA$_\$(3)_{M_3(\mathbb{Z})}$ by Lemma 4.3. By using additional states, we can obtain an equivalent XBHVA$(3)_{M_3(\mathbb{Z})}$ without end-marker by Lemma 4.1.

We are going to show that the inclusion is proper by constructing a NBHVA$(3)_{S_3(1)}$ $\mathcal{V}$ recognizing the unary nonregular language UPOW′ $= \{a^{n+2^n} | n \geq 1\}$. The state transition diagram of $\mathcal{V}$ is given in Figure 4.4. Starting with the initial vector $(1 \;\; 1 \;\; 1)$, $\mathcal{V}$ multiplies the vector with matrix A_1 when reading each a. The idea is to add the first and second entries together repeatedly to obtain powers of 2, so that after reading k symbols the value of the vector is equal to $\begin{pmatrix} 2^k & 2^k & 1 \end{pmatrix}$. $\mathcal{V}$ nondeterministically guesses n and starts decrementing the first entry from that point on by multiplying the vector with the matrix A_2 which fixes the second entry to 1 immediately. At the end of the computation, the value of the vector is equal to $(1 \;\; 1 \;\; 1)$ if and only if the input string is of the form a^{n+2^n} for some n.

We conclude the result since no XkBCA can recognize the language UPOW′ by Fact 2.2.

□



$$A_1 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad A_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ -1 & 1 & 1 \end{pmatrix}$$

Figure 4.4. State transition diagram of $\mathcal{V}$ recognizing UPOW′

If we further restrict the matrices to have entries from the set $\{-1, 0, 1\}$, then we obtain the following result.

Theorem 4.11. $\bigcup_k \mathfrak{L}(\mathrm{X}k\mathrm{BCA}) \subsetneq \bigcup_k \mathfrak{L}(\mathrm{XBHVA}(k+1)_{S_{k+1}(1)})$ *where* $\mathrm{X} \in \{\mathrm{D,N,1D,1N}\}$.

Proof. Let us simulate a given XkBCA by a XBHVA$(k+1)_{S_{k+1}(1)}$ $\mathcal{V}$. Let $(1 \;\; 1 \;\; \cdots \;\; 1)$ be the initial vector of $\mathcal{V}$. The $k+1$'st entry of the vector will remain unchanged throughout the computation, which will allow the counter updates. At each step of the computation, $\mathcal{V}$ will multiply the vector with the appropriate matrix $A \in T$ where $T \subseteq S_{k+1}(1)$ is the set of all $(k+1) \times (k+1)$ matrices corresponding to possible counter updates. Since each counter can be decremented, incremented or left unchanged, $|T| = 3^k$. All matrices will have the property that $A[i,i] = 1$ and $A[k+1,k+1] = 1$. When the i'th counter is incremented and decremented, then $A[k+1,i] = 1$ and $A[k+1,i] = -1$, respectively. At the end of the computation, the input will be accepted if the vector is equal to $(1 \;\; 1 \;\; \cdots \;\; 1)$, which happens iff all counters have value 0.

The inclusions are proper by the witness languages $\mathtt{MPAL}_2$, which can be recognized by a DBHVA$(2)_{S_2(1)}$ by Example 4.6 for the deterministic case and UPOW′, which can be recognized by a NBHVA$(3)_{S_3(1)}$ by Theorem 4.10. □

4.4.2. Non-blind Counter Automata

The fact that the individual entries of the vector cannot be checked prevents us from simulating a non-blind counter automaton by a HVA. Since 1D2CA can recognize any recursively enumerable language, we focus on the real-time case.

We are going to show that in the real-time deterministic case, counter automata and blind homing vector automata are incomparable. First, we give a characterization for DHVA(k)s when the alphabet is unary.

Theorem 4.12. *For any k, all languages over $\Sigma = \{a\}$ accepted by a* DHVA(k) *are regular.*

Proof. Let L be a unary language accepted by a DHVA(k) $\mathcal{V}$ and let v be the initial vector of $\mathcal{V}$. We are going to construct a DFA recognizing L to prove that L is regular. We assume that L is infinite and make the following observation. Since $\mathcal{V}$ has finitely many states, at least one of the accept states of $\mathcal{V}$ will be accepting more than one string. Let w_1 and w_2 be the shortest strings accepted by an accept state q_a with $|w_1| < |w_2|$. When accepting w_1 and w_2, $\mathcal{V}$ is in state q_a and the value of the vector is equal to v. After reading w_2, $\mathcal{V}$ is in the same configuration as it was after reading w_1 and this configuration will be repeated inside a loop of $|w_2| - |w_1| = p$ steps. Therefore, we can conclude that all strings of the form $a^{|w_1|+lp}$ for some positive integer l will be accepted by q_a.

Between consecutive times q_a accepts a string, some other strings may be accepted by some other accept states. Let u be a string accepted by q_b with $|w_1| < |u| < |w_2|$. Then all strings of the form $a^{|u|+lp}$ for some positive integer l will be accepted by q_b since every time $\mathcal{V}$ enters the accepting configuration at state q_a, $\mathcal{V}$ will enter the accepting configuration at state q_b after $|u| - |w_1|$ steps. The same reasoning applies to any other accepting configuration inside the loop.

Now, let us construct a DFA $\mathcal{F}$ accepting L. $\mathcal{F}$ has $|w_1|+1+(p-1)$ states. The first $|w_1|+1$ states correspond to the strings of length at most $|w_1|$ and the state $q_{|w|}$ is an accept state for all $w \in L$ that is of length at most $|w_1|$. $q_{|w_1|}$ and the next $p-1$ states $q_{l_2}, \ldots, q_{l_p}$ stand for the configuration loop. States corresponding to accepting configurations inside the loop are labeled as accept states.

The transitions of the $\mathcal{F}$ are as follows:

$$\begin{aligned} \delta(q_i, a) &= q_{i+1} \text{ for } i = 0, \ldots, |w_1| - 1 \\ \delta(q_{|w_1|}, a) &= q_{l_2} \\ \delta(q_{l_i}, a) &= q_{l_{i+1}} \text{ for } i = 2, \ldots, p-1 \\ \delta(q_{l_p}, a) &= q_{|w_1|} \end{aligned}$$

Since L can be recognized by a DFA, L is regular. We conclude that any unary language accepted by a DHVA(k) is regular. □

Theorem 4.13. $\bigcup_k \mathfrak{L}(\text{DBHVA}(k))$ *and* $\bigcup_k \mathfrak{L}(\text{D}k\text{CA})$ *are incomparable.*

Proof. We know that $\texttt{MPAL}_2 = \{w\#w^r | w \in \{0,1\}^*\}$ can be recognized by a DBHVA(2) by Example 4.6. In [22], it is proven that no deterministic counter machine with k counters operating in time $O(2^{n/k})$ can recognize $\texttt{MPAL}_2$. Since we are working with real-time machines, this result applies to our case.

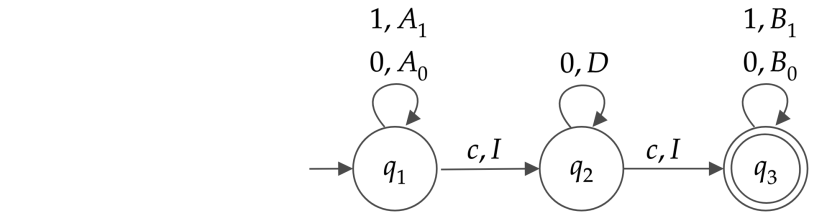
On the other hand, it is known that the nonregular unary language $\texttt{UGAUSS} = \{a^{n^2+n} | n \in \mathbb{N}\}$ can be recognized by a D2CA [9]. By Theorem 4.12, we know that BHVA(k)'s and inherently DBHVA(k)'s can recognize only regular languages in the unary case. Hence, we conclude that the two models are incomparable.

□

For the nondeterministic case we can state the following result.

Theorem 4.14. $\bigcup_k \mathfrak{L}(\text{NBHVA}(k)) \not\subseteq \bigcup_k \mathfrak{L}(\text{N}k\text{BCA})$.

Proof. It is known that no NkBCA can recognize the language $\texttt{BIN} = \{wc0^{e_2(w)}cw^rc | w \in 1\{1,0\}^*\}$ [21] where $e_2(w)$ is the integer representation of the binary number represented by w, for any k. A NBHVA(3) $\mathcal{V}$ recognizing the language BIN is given in Figure 4.5.



$$A_0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} \quad A_1 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} \quad D = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad B_0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/2 \end{pmatrix} \quad B_1 = \begin{pmatrix} 1 & 0 & -1/2 \\ 0 & 1 & 0 \\ 0 & 0 & 1/2 \end{pmatrix}$$

Figure 4.5. State transition diagram of $\mathcal{V}$ recognizing BIN

The initial vector of $\mathcal{V}$ is $(1 \;\; 0 \;\; 0)$. The idea is to use base-2 encoding to store $e_2(w)$ in the second and the third entries of the vector with the help of the matrices A_0 and A_1. While reading the 0 sequence, the second entry is decremented using the matrix D. While reading the last sequence between two c's, the inverses of the matrices used for base-2 encoding is used to check whether the sequence corresponds to the string w^r. □

4.5. Relationship with Extended Finite Automata

In this section, we will exploit a relationship between 1NBHVA(k)'s and the extended finite automata over free groups to demonstrate the power of homing vector automata.

The two models seem to be linked in the case of finite automata over matrix groups, as the register is multiplied with a matrix at each step of the computation. Let us emphasize that the two models are different in the following sense. In a homing vector automaton, there is an initial vector v, and the accepted strings are those which label a computation path along which the product of the sequence of matrices on the transitions is a matrix A, such that $v = vA$. In the most general setting, the set of transition matrices belongs to the semigroup of rational matrices. In an accepting computation, the product of the matrices A belongs to the stabilizer subsemigroup of the set of rational matrices with respect to v. In contrast, in an extended finite automaton over a matrix group, accepting computations are those in which $A = I$. In that sense, one-way nondeterministic blind homing vector automata can be seen as akin to what someone who wanted to define a version of extended finite automata associated with general matrix semigroups, rather than groups, would come up with. In fact, the homing vector automaton can be seen as a special case of the rational semigroup automaton which is defined in [69] as follows:

Let M be a monoid. An M-automaton is said to be *with targets* if it is equipped with two subsets $I_0, I_1 \subseteq M$ called the initial set and the terminal set respectively. An input string $w \in \Sigma^*$ is accepted by the automaton if there exists a computation from the initial state to some accepting state such that $x_0 x \in I_1$, where $x_0 \in I_0$ and $x \in M$ is the content of the register of the machine after the reading of w. In the case that I_0 and I_1 are rational subsets of M, the model is called *rational monoid automaton* defined by M [35, 69].

Rational semigroup automata are defined analogously by taking M as a semigroup instead of a monoid.

Note that the family of languages accepted by rational monoid automata where $I_0 = I_1 = \{1\}$ coincides with the set of languages recognized by ordinary M-automata. Letting $I_0 = I_1 = v$ where v is the initial vector of a homing vector automaton and M to be a semigroup of matrices, one obtains homing vector automata.

We start by looking at the case of 2×2 matrices. Recall from Subsection 3.2.2 that $\mathbf{F}_2$ admits a representation by 2×2 matrices. In particular, the group generated by the matrices

$$M_a = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad M_b = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$$

is isomorphic to $\mathbf{F}_2$.

Using the matrix representation of $\mathbf{F}_2$, any $\mathbf{F}_2$-automaton can be simulated by a suitably defined homing vector automaton that is of dimension 2, blind, nondeterministic, and one-way. The proof is due to Flavio D'Alessandro and can be found in [16].

Theorem 4.15. $\mathfrak{L}(\mathbf{F}_2) \subseteq \mathfrak{L}(\text{1NBHVA(2)})$.

This allows us to draw the following conclusion about the class of languages recognized by 1NBHVA(2)'s.

Corollary 4.16. *The family of context-free languages is included in* $\mathfrak{L}(\text{1NBHVA(2)})$.

Proof. Since $\mathfrak{L}(\mathbf{F}_2) = \mathsf{CF}$ by Fact 3.3, the result then follows by Theorem 4.15. □

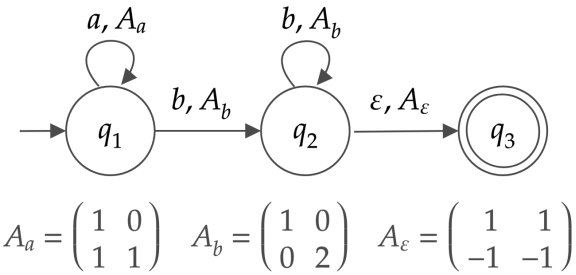
In the proof of Theorem 4.15, the matrices used for the simulation of an $\mathbf{F}_2$-automaton belong to $SL(2, \mathbb{Z})$. In the following theorem, we show that 1NBHVAs are more powerful than the corresponding monoid automata, when the matrices are restricted to the monoid $M_2(\mathbb{Z})$.

Theorem 4.17. $\mathfrak{L}(M_2(\mathbb{Z})) \subsetneq \mathfrak{L}(\text{1NBHVA(2)}_{M_2(\mathbb{Z})})$.

Proof. In Theorem 3.10, it is proven that $M_2(\mathbb{Z}) = \mathsf{CF}$ and by Corollary 4.16, the inclusion follows.

Now we are going to prove that the inclusion is proper. Let us construct a 1NBHVA(2)$_{M_2(\mathbb{Z})}$ $\mathcal{V}$ recognizing the non-context-free language $\texttt{POW}_\texttt{r} = \{a^{2^n}b^n | n \geq 0\}$. The state diagram of $\mathcal{V}$ is given in Figure 4.6.

Figure 4.6. State transition diagram of $\mathcal{V}$ recognizing $\texttt{POW}_\texttt{r}$



The initial vector of $\mathcal{V}$ is $v = (1 \;\; 1)$. While reading a in q_1, $\mathcal{V}$ multiplies its vector with the matrix A_a. It moves to q_1 when it scans the first b and multiplies its vector with the matrix A_b as it reads each b. $\mathcal{V}$ multiplies its vector with the matrix A_ε and moves to q_2.

When the vector is multiplied by A_a, the first entry of the vector is increased by 1 and when the vector is multiplied by A_b, the second entry of the vector is multiplied by 2. Hence, after reading an input string of the form a^ib^j, the vector is equal to $(i+1 \;\; 2^j)$, as a result of the multiplication by the matrix product ${A_a}^i{A_b}^j$. After multiplication by A_ε, the second entry of the vector is subtracted from the first entry and this value is stored in both entries of the vector. The value of the final vector is equal to $(1 \;\; 1)$ iff $i + 1 - 2^j = 1$. Hence, the accepted strings are those which are of the form $a^{2^j}b^j$. □

For 3×3 matrices, we don't have a comparability result among the classes of languages recognized by the two models. The major limitation lies in the fact that the limits of finite automata over 3×3 matrix groups are not known. Furthermore, we don't know how to directly simulate a finite automaton overa group of 3×3 matrices with a 1NBHVA(3). Nevertheless, let us note that 1NBHVA(3)s defined with matrices from $M_3(\mathbb{Z})$ can recognize any language recognized by 1NkBCAs as we showed in Theorem 4.10, whereas it is an open question whether $\mathfrak{L}(M_3(\mathbb{Z}))$ includes the class of languages recognized by 1N3BCAs.

Next we are going to demonstrate the power of 1NBHVA(k)s for $k \geq 4$. Recall that $\mathfrak{L}(\mathbf{F}_2 \times \mathbf{F}_2) = \mathfrak{L}(SL(4,\mathbb{Z}))$ by Theorem 3.17. The proof idea of Theorem 3.17 is to embed two copies of $\mathbf{F}_2$ into 4×4 matrices. Combining this idea with the proof of Theorem 4.15, one can show that 1NBHVA(4)s recognize any recursively enumerable language. Since the computation of a 1NBHVA(k) involves vector matrix multiplications by computable numbers, it can be simulated by a Turing machine and it follows that any language recognized by a 1NBHVA(k) is recursively enumerable. The proof details are omitted here and can be found in [16].

Theorem 4.18. $\mathsf{RE} = \mathfrak{L}(\text{1NBHVA(k)})$ *for* $k \geq 4$.

4.6. Real-time Homing Vector Automata

In the previous section, we have seen that allowing one-way access to the input tape raises nondeterministic blind homing vector automata of small vector dimension to Turing equivalence. For this reason, we will be focusing on real-time input in this section.

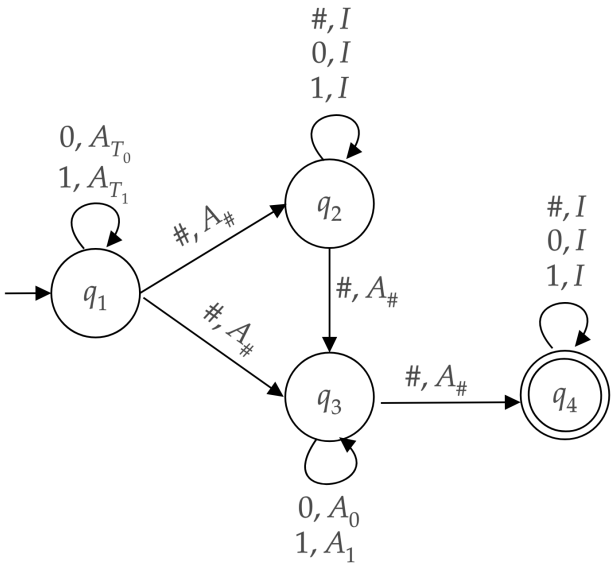
Let us start with an example. We show that by allowing nondeterminism, it is possible to recognize an NP-complete language in real-time and with matrices belonging to set $S_5(1)$. SUBSETSUM is the NP-complete language which is the collection of all strings of the form $t\#a_1\#...\#a_n\#$, such that t and the a_i's are numbers in binary notation ($1 \leq i \leq n$), and there exists a set $I \subseteq \{1,...,n\}$ satisfying $\sum_{i\in I} a_i = t$, where $n > 0$. We define

$$\texttt{SUBSETSUM}_r = \{t^r\#a_1^r\#...\#a_n^r\# \mid \exists I \subseteq \{1,...,n\} \text{ s.t. } \sum_{i\in I} a_i = t\}$$

in which the binary numbers appear in reverse order. It is obvious that $\texttt{SUBSETSUM}_\texttt{r} \in$ NP, since SUBSETSUM $\in$ NP. It is possible to reduce SUBSETSUM to $\texttt{SUBSETSUM}_\texttt{r}$ in polynomial time by reversing the binary numbers that appear in the input. Therefore, we can conclude that $\texttt{SUBSETSUM}_\texttt{r}$ is NP-complete.

Example 4.19. $\texttt{SUBSETSUM}_\texttt{r} \in \mathfrak{L}(\text{NBHVA(5)})$.

We construct a NBHVA$(5)_{M_5(1)}$ $\mathcal{V}$ recognizing $\texttt{SUBSETSUM}_\texttt{r}$. The state transition diagram of $\mathcal{V}$ recognizing $\texttt{SUBSETSUM}_\texttt{r}$ is given in Figure 4.7.



$$A_{T_0} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad A_{T_1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad A_{\#} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

$$A_0 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad A_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 4.7. State transition diagram of $\mathcal{V}$ recognizing $\texttt{SUBSETSUM}_\texttt{r}$

The idea of this construction is to read the binary numbers in the string to entries of the vector, and to nondeterministically select the set of numbers that add up to t. We let the initial vector equal $(0 \;\; 0 \;\; 1 \;\; 1 \;\; 1)$. We first encode t to the first entry of the vector as follows: While scanning the symbols of t, $\mathcal{V}$ multiplies the vector with the matrix A_{T_0} (resp. A_{T_1}) for each scanned 0 (resp. 1). The powers of 2 required for the encoding are obtained by adding the third and fourth entries, which always contain identical numbers, to each other, creating the effect of multiplication by 2.

When $\mathcal{V}$ reads a #, $\mathcal{V}$ multiplies the vector with the matrix $A_{\#}$ which subtracts the second entry from the first entry and resets the second entry back to 0, and the third and fourth entries back to 1. In the rest of the computation, $\mathcal{V}$ nondeterministically decides which a_i's to subtract from the first entry. Each selected a_i is encoded using the same technique into the second entry of the vector. While scanning the symbols of a_i, $\mathcal{V}$ multiplies the vector with the matrix A_0 (resp. A_1) for each scanned 0 (resp. 1).

$\mathcal{V}$ chooses another a_j if it wishes, and the same procedure is applied. At the end of the input, $\mathcal{V}$ accepts if the vector is equal to $(0 \;\; 0 \;\; 1 \;\; 1 \;\; 1)$, which requires that the first entry of the vector is equal to 0. This is possible iff there exists a set of a_i's whose sum add up to t.

4.6.1. Comparisons Among the Different Versions

We start by comparing the deterministic blind and non-blind versions of our model.

Theorem 4.20. $\bigcup_k \mathfrak{L}(\text{DBHVA}(k)) \subsetneq \bigcup_k \mathfrak{L}(\text{DHVA}(k))$.

Proof. It is obvious that any DBHVA(k) can be simulated by a DHVA(k). We are going to prove that the inclusion is proper by the witness language $\texttt{SUM} = \{a^n b^{a_1} a^{a_2} | n = a_1 \text{ or } n = a_1 + a_2, a_1 \geq 1, a_2 \geq 1\}$. Let us first construct a DHVA$(2)_{S_2(1)}$ $\mathcal{V}$ recognizing SUM. The state transition diagram of $\mathcal{V}$ is given in Figure 4.8. The idea is to simulate a counter with the help of the matrices. Starting with the initial vector $(1 \;\; 1)$, $\mathcal{V}$ multiplies the vector with the matrix A_+ for each a it reads before the b's, incrementing the first entry of the vector with each such multiplication. After finishing reading the first segment of a's, $\mathcal{V}$ multiplies the vector with the matrix A_-, decrementing the first entry of the vector for each b.

$\mathcal{V}$ checks the current value of the vector for equality to $(1 \;\; 1)$ when reading the first a. If the equality is detected, it is the case that $n = a_1$, and $\mathcal{V}$ multiplies the vector with the identity matrix at each step for the rest of the computation. If that is

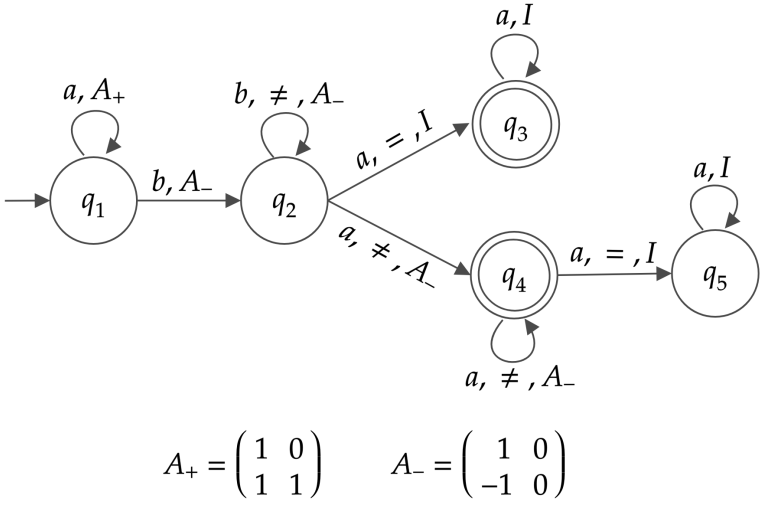


Figure 4.8. State transition diagram of $\mathcal{V}$ recognizing SUM

not the case, $\mathcal{V}$ continues to multiply the vector with matrix A_- for each a after the b's. The value of the vector will be equal to (1 1) at the end of the computation if and only if $n = a_1$ or $n = a_1 + a_2$.

Note that SUM can be also recognized by a DHVA(1) by using the one-dimensional matrices $A_+ = (2)$ and $A_- = (\frac{1}{2})$.

Now we are going to show that SUM cannot be recognized by any DBHVA(k). Suppose for a contradiction that L is recognized by some DBHVA(k) $\mathcal{V}'$. After reading a prefix of a's, the computation of $\mathcal{V}'$ on a sufficiently long suffix of b's will go through a sequence of states, followed by a state loop. Suppose that $\mathcal{V}'$ is in the same state after reading two different strings $a^n b^m$ and $a^n b^n$, $m < n$. Now consider the strings $u = a^n b^m a^{n-m} \in$ SUM and $w = a^n b^n a^{n-m} \in$ SUM. After reading any one of these strings, $\mathcal{V}'$ should be in the same accept state, and the vector should be at its initial value. Assume that the strings in question are both extended with one more a. Since the same vector is being multiplied with the same matrix associated with the same state during the processing of that last a, it is not possible for $\mathcal{V}'$ to give different responses to $a^n b^n a^{n-m+1}$ and $a^n b^m a^{n-m+1}$. Noting that $a^n b^n a^{n-m+1} \in$ SUM, whereas $a^n b^m a^{n-m+1} \notin$ SUM, we conclude that SUM cannot be recognized by any DBHVA(k). □

In the following theorem, we show that nondeterministic real-time homing vector automata are more powerful than their deterministic versions, both in the blind and nonblind cases.

Theorem 4.21. *(i)* $\bigcup_k \mathfrak{L}(\text{DBHVA}(k)) \subsetneq \bigcup_k \mathfrak{L}(\text{NBHVA}(k))$.
(ii) $\bigcup_k \mathfrak{L}(\text{DHVA}(k)) \subsetneq \bigcup_k \mathfrak{L}(\text{NHVA}(k))$.

Proof. It is obvious that the deterministic models can be simulated by the nondeterministic models. The inclusion is proper since $\texttt{UPOW}'$ can be recognized by a NBHVA(2) by Theorem 4.9 and every unary language recognized by a DHVA(k) is regular by Theorem 4.12. □

A language L is in class $\mathsf{TISP}(t(n), s(n))$ if there is a deterministic Turing machine that decides L within $t(n)$ time and $s(n)$ space where n is the length of the input. Since the numbers in the vector can grow by at most a fixed number of bits in each multiplication, a Turing machine simulating a DHVA(k) requires only linear space [9]. Since the numbers in the vector can have length $O(n)$, whereas the matrix dimensions and entries are independent of the input length n, multiplication of a vector and a matrix requires $O(n)$ time for each input symbol. We can conclude that $\bigcup_k \mathfrak{L}(\text{DHVA}(k)) \subseteq \mathsf{TISP}(n^2, n)$.

4.6.2. A Hierarchy Result

We will now establish a hierarchy result on real-time deterministic homing vector automata based on the dimension of the vector, when the matrix entries belong to a restricted set.

Theorem 4.22. *Then* $\mathfrak{L}(\text{DHVA}(k))_{S_k(m)} \subsetneq \mathfrak{L}(\text{DHVA}(l))_{S_l(m)}$ *for* $l > (km)^k$.

Proof. Using the generalized Stern-Brocot encoding, we showed that it is possible to recognize $\texttt{MPAL}_1 = \{w\#w^r | w \in \{a_1, a_2, \ldots, a_l\}^*\}$ by a DBHVA$(l)_{S_l(1)}$ in Exercise 4.6.

We are now going to show that $\texttt{MPAL}_\texttt{1} \notin \mathfrak{L}(\text{DHVA}(k))_{S_k(m)}$ for $l > (km)^k$. We first note that the value of any entry of a vector of size k can be at most $m^{n+1}k^n$ after reading n symbols. This is possible by letting the initial vector have m in all entries, and multiplying the vector with the matrix with all entries equal to m at each step. Similarly, the smallest possible value of an entry is $-m^{n+1}k^n$, and so the number of possible different values for a single entry is $2m^{n+1}k^n + 1$. If the machine has s states, $s(2m^{n+1}k^n + 1)^k$ is an upper bound for the number of different reachable configurations after reading n symbols. Since there are l^n strings of length n when the alphabet consists of l symbols, for large n and $l > (km)^k$, the machine will end up in the same configuration after reading two different strings u and w. This will cause the strings $u\#w^r$ and $w\#u^r$ which are not in $\texttt{MPAL}_\texttt{1}$ to be accepted by the machine. Therefore, we conclude that $\texttt{MPAL}_\texttt{1} \notin \mathfrak{L}(\text{DHVA}(k))_{S_k(m)}$.

Since a vector automaton with a larger vector size can trivially simulate a vector automaton with a smaller vector size, this result applies to our case. □

4.6.3. Closure Properties

In this section, we examine the closure properties of the class of languages recognized by real-time homing vector automata. We start with a lemma which will be useful in our proofs. The languages mentioned below are from [6].

Lemma 4.23. *(i)* $\texttt{UNION} = \{a^n b^n | n \geq 0\} \cup \{a^n b^{2n} | n \geq 0\} \notin \bigcup_k \mathfrak{L}(\text{DHVA}(k))$.
(ii) $\texttt{L}_\texttt{bab} = \{b^n (a^n b^n)^k | n, k \geq 1\} \notin \bigcup_k \mathfrak{L}(\text{DHVA}(k))$.
(iii) $\texttt{IJK} = \{a^i b^j c^k | i \neq j \text{ or } j > k\} \notin \bigcup_k \mathfrak{L}(\text{DHVA}(k))$.
(iv) $\texttt{UNION}_\texttt{c} = \{a^n b^n | n \geq 0\} \cup \{a^n b^{2n} c | n \geq 0\} \notin \bigcup_k \mathfrak{L}(\text{DHVA}(k))$.

Proof. We can show all these languages to be unrecognizable by DHVAs by applying the following common reasoning. Assume that the language L in question is recognized by some DHVA(k) $\mathcal{V}$. Since there are finitely many states, one of the states of $\mathcal{V}$ will end up accepting more than one member of the language. For each language, we will focus on two such members u and v. Note that $\mathcal{V}$ is in the same configuration (since

it has also returned to its initial vector) after reading both u and v. We then append another string x to both strings, selected so that $ux \in L$ and $vx \notin L$. The responses of $\mathcal{V}$ to the ux and vx has to be identical, since it will have returned to the same configuration after processing both strings. We conclude that $\mathcal{V}$ cannot distinguish between these two strings, and therefore that $L \notin \bigcup_k \mathfrak{L}(\text{DHVA}(k))$. All that remains is to provide the strings u, v, and x for the languages in the statement of the lemma. In the following, $i, j > 1$ and $i \neq j$.

(i) $u = a^i b^i$, $v = a^j b^j$, and $x = b^i$.

(ii) $u = b^i a^i b^i$, $v = b^j a^j b^j$ and $x = a^i b^i$.

(iii) $u = a^i b^i c$, $v = a^j b^j c$, and $x = c^{j-1}$ for $i > j$.

(iv) $u = a^i b^i$, $v = a^j b^j$, and $x = b^i c$.

□

Let us note that it is possible to recognize the languages mentioned in the proofs with DHVA(k)'s of smaller vector size when the vector entries are not restricted to be integers.

Theorem 4.24. *(i)* $\bigcup_k \mathfrak{L}(\text{DHVA}(k))$ *is closed under the following operations:*

(a) intersection with a regular set

(ii) $\bigcup_k \mathfrak{L}(\text{DHVA}(k))$ *is not closed under the following operations:*

(a) union

(b) concatenation

(c) intersection

(d) star

(e) homomorphism

(f) reversal

(g) complementation

Proof.

(i) (a) Let $L_{\mathcal{V}}$ be recognized by a DHVA(k) $\mathcal{V} = (Q_1, \Sigma_1, \mathrm{M}, \delta_1, q_1, Q_{a_1}, v)$ and $L_{\mathcal{F}}$ be a language recognized by a finite state automaton $\mathcal{F} = (Q_2, \Sigma_2, \delta_2, q_2, Q_{a_2})$. Let us construct a DHVA(k) $\mathcal{V}' = (Q, \Sigma, \mathrm{M}, \delta, q_1, Q_a, v)$ recognizing $L = L_{\mathcal{V}} \cap L_{\mathcal{M}}$. $\mathcal{V}'$ keeps track of the vector and the current state of $\mathcal{V}$ as well as the current state of $\mathcal{F}$. Let $Q' = Q_1 \times Q_2$ be the state set of $\mathcal{V}'$ and $\Sigma = \Sigma_1 \cup \Sigma_2$. For each $(q_i, q_j) \in Q$, $\sigma \in \Sigma$ and $\omega \in \Omega$, $\delta((q_i, q_j), \sigma, \omega) = ((q_i', q_j'), A)$ where $\delta_1(q_i, \sigma, \omega) = (q_i', A)$ and $\delta_2(q_j, \sigma) = q_j'$. q_1 is the pair (q_1, q_2) and Q_a is the set of pairs of states where both of the states are accept states of $\mathcal{V}$ or $\mathcal{F}$. We obtain a DHVA(k) $\mathcal{V}'$ recognizing L.

(ii) (a) Let $L_1 = \{a^n b^n | n \geq 0\}$ and $L_2 = \{a^n b^{2n} | n \geq 0\}$. L_1 and L_2 can be recognized by a DBHVA(2) which simulates a deterministic blind one-counter automaton whereas $L_1 \cup L_2 = \texttt{UNION}$ cannot be recognized by any DHVA(k) for any k by Lemma 4.23.

(b) For the languages $L_1 = \{a^n b^n | n \geq 0\}$ and $L_2 = \{a^n b^{2n} | n \geq 0\}$, $L_1 L_2 \cap a^* b^* = \texttt{UNION}$, which cannot be recognized by any DHVA(k) for any k by Lemma 4.23 and Part i.a) of this theorem.

(c) Let $\mathtt{L_1} = \{b^+ (a^n b^n)^* | n \geq 1\}$ and $\mathtt{L_2} = \{(b^n a^n)^* b^+ | n \geq 1\}$. Both L_1 and L_2 can be recognized by DHVA(2)s which simulate deterministic one-counter automata, whereas $L_1 \cap \mathtt{L_2} = \mathtt{L_{bab}} = \{b^n (a^n b^n)^k | n, k \geq 1\}$ cannot be recognized by any DHVA(k) for any k by Lemma 4.23.

(d) Let $L = \{a^n b^n | n \geq 0\} \cup \{c a^n b^{2n} | n \geq 0\}$. A DBHVA(2) $\mathcal{V}$ recognizing L branches into one of two computation paths depending on the first scanned symbol σ_1. If $\sigma_1 = a$, $\mathcal{V}$ simulates a deterministic blind one-counter automaton recognizing $\{a^{n-1} b^n | n \geq 0\}$ and if $\sigma_1 = c$, $\mathcal{V}$ simulates a deterministic blind one-counter automaton recognizing $\{a^n b^{2n}\}$. Now suppose $L^* \in \bigcup_k \mathfrak{L}(\mathrm{DHVA}(k))$. Then $L' = L^* \cap \{c a^i b^j | i, j \geq 0\} = \{c a^n b^n | n \geq 0\} \cup \{c a^n b^{2n} | n \geq 0\} \in \bigcup_k \mathfrak{L}(\mathrm{DHVA}(k))$. A DHVA($k$) recognizing L' can be easily modified to obtain a DHVA(k) recognizing the language $\texttt{UNION} = \{a^n b^n | n \geq 0\} \cup \{a^n b^{2n} | n \geq 0\}$, which is not in $\mathfrak{L}(\mathrm{DHVA}(k))$ by Lemma 4.23.

(e) Let $L = \{a^n b^n | n \geq 0\} \cup \{ca^{n-1}b^{2n} | n \geq 0\}$. A DBHVA($k$) recognizing L works similarly to the one in part d). Now consider the homomorphism h such that $h(a) = a$, $h(b) = b$ and $h(c) = a$. $h(L) = \{a^n b^n | n \geq 0\} \cup \{a^n b^{2n}\} = \texttt{UNION}$, which cannot be recognized by any DHVA(k) for any k by Lemma 4.23.

(f) Let $L = \{b^n a^n | n \geq 0\} \cup \{cb^{2n}a^n | n \geq 0\}$. A DBHVA($k$) recognizing L works similarly to the one in part d). Now consider the reverse of L, $\texttt{UNION}_\texttt{c} = \{a^n b^n | n \geq 0\} \cup \{a^n b^{2n} c | n \geq 0\}$, which cannot be recognized by any DHVA(k) for any k by Lemma 4.23.

(g) Consider $L = \{a^m b^m c^n | 0 \leq m \leq n\}$, which can be recognized by a DHVA(3). $\bar{L} \cap \{a^i b^j c^k | i, j, k \geq 0\} = \{a^i b^j c^k | i \neq j \text{ or } j > k\} = \texttt{IJK}$ cannot be recognized by any DHVA(k) by Lemma 4.23.

□

The set of languages recognized by real-time nondeterministic homing vector automata is closed under union, star and concatenation The constructions are fairly simple and omitted.

Theorem 4.25. *(i)* $\bigcup_k \mathfrak{L}(\text{DBHVA}(k))$ *is closed under the following operations:*

(a) intersection

(ii) $\bigcup_k \mathfrak{L}(\text{DBHVA}(k))$ *is not closed under the following operations:*

(a) union

(b) concatenation

(c) star

(d) homomorphism

(e) reversal

(f) complementation

Proof.

(i) (a) Let $L_{\mathcal{V}_1}$ and $L_{\mathcal{V}_2}$ be recognized by DBHVA(k_1) $\mathcal{V}_1 = (Q_1, \Sigma_1, \delta_1, q_1, Q_{a_1}, v_1)$ and DBHVA(k_2) $\mathcal{V}_2 = (Q_2, \Sigma_2, \delta_2, q_2, Q_{a_2}, v_2)$, respectively. Let us construct

a DBHVA(k) $\mathcal{V} = (Q, \Sigma, \delta, q_1, Q_a, v)$ recognizing $L = L_{\mathcal{V}_1} \cap L_{\mathcal{V}_2}$ where $k = k_1 + k_2$. Let $Q = Q_1 \times Q_2$ be the state set of V and $\Sigma = \Sigma_1 \cup \Sigma_2$. For each $(q_i, q_j) \in Q$ and $\sigma \in \Sigma$, $\delta((q_i, q_j), \sigma) = ((q_i', q_j'), A)$, where $\delta_1(q_i, \sigma) = (q_i', A_1)$, $\delta_2(q_j, \sigma, \omega) = (q_j', A_2)$ and A is a $k \times k$ block diagonal matrix with A_1 and A_2 on its diagonal. q_1 is the pair (q_1, q_2), and Q_a is the set of pairs of states where both of the states are accept states of $\mathcal{V}_1$ or $\mathcal{V}_2$. The initial vector v of $\mathcal{V}$ is of the form $(v_1 \quad v_2)$ and has dimension k. $\mathcal{V}$ keeps track of the current states and the current values of both vectors by simultaneously multiplying its vector with the appropriate matrices. Since the computation is blind, the value of the vector is checked only at the end of the computation, and an input string is accepted if the vector is equal to its initial value.

(ii) The proofs for the non-blind version also apply here. The proof for part (f) follows from the fact that $\bigcup_k \mathfrak{L}(\text{DBHVA}(k))$ is closed under intersection but not union.

□

The set of languages recognized by real-time nondeterministic blind homing vector automata is closed under union and intersection. The construction for union is straightforward, and the construction for intersection is identical to the deterministic case.

4.6.4. Stateless computation

Given two strings, a finite automaton is said to separate them if it accepts one and rejects the other. Introduced by Goralčík and Koubek [70], the string separation problem asks for the minimum number of states needed for accomplishing this task. String separation by homing vector automata and vector automata have been investigated in [17]. It turns out that a homing vector automaton needs at least two states to separate any pair of strings, regardless of the dimension of the vector. We are therefore motivated to examine the limitations imposed by statelessness on homing vector automata in more detail.

Stateless machines [71–73] have been investigated by many researchers, motivated by their relation to membrane computing and P systems [74], which are stateless models inspired from biology. While vector automata can simulate their classical states in their vectors by using longer vectors, this is not the case for homing vector automata.

Our study on stateless homing vector automata yields a characterization for the class of languages recognized by stateless real-time deterministic FAMs without equality (0-DFAMW) [6]. It turns out that a language is recognized by a 0-DFAMW iff it is commutative and its Parikh image is the set of nonnegative solutions to a system of linear homogeneous Diophantine equations. When the computation is nondeterministic, then any language recognized by a stateless real-time nondeterministic FAM without equality is commutative. We conclude by providing some further examples and observations about language recognition power of stateless homing vector automata.

4.6.4.1. Observations. The limitation of having a single state for homing vector automata leads to the acceptance of the string xx, whenever the string x is accepted. This is true since further repetition of the same input naturally carries the vector through a trajectory that ends up in its initial value. Based on this observation, we can list the following consequences:

For $X \in \{\text{DB,D,NB,N}\}$,

- If string x is accepted by a 0-XHVA $\mathcal{V}$, then any member of $\{x\}^*$ is also accepted by $\mathcal{V}$.
- If all members of a language L are accepted by a 0-XHVA $\mathcal{V}$, then any string in L^* is also accepted by $\mathcal{V}$.
- If language L is recognized by a 0-XHVA, then $L = L^*$.
- 0-XHVAs cannot recognize any finite language except L_ε.

We can further make the following observation for deterministic models.

Lemma 4.26. *If the strings w_1 and w_1w_2 are accepted by a* 0-XHVA *V where $X \in$* {DB,D}*, then the string w_2 is also accepted by $\mathcal{V}$.*

Proof. After reading w_1, the value of the vector is equal to its initial value. Since w_1w_2 is also accepted by $\mathcal{V}$, reading w_2 results in acceptance when started with the initial vector. □

For the unary case we have the following.

Theorem 4.27. *If the strings a^i and a^j $(1 < i < j)$ are accepted by a* 0-XHVA *$\mathcal{V}$ where $X \in$* {DB,D}*, then the string $a^{\gcd(i,j)}$ is also accepted by $\mathcal{V}$.*

Proof. It is well known that for any positive integers i, j, there are two integers l_i and l_j such that $il_i + jl_j = \gcd(i, j)$. Assume that l_i is positive and l_j is non-positive. (The other case is symmetric.) Note that $il_i \geq j(-l_j)$. The strings $a^{j(-l_j)}$ and a^{il_i} are accepted by H. By Lemma 4.26, the string $a^{il_i - j(-l_j)}$, which is $a^{\gcd(i,j)}$, is also accepted by $\mathcal{V}$.

□

Corollary 4.28. *If the strings a^i and a^j $(1 < i < j)$ are accepted by a* 0-XHVA *H where $X \in$* {DB,D} *and* $\gcd(i, j) = 1$*, then $\mathcal{V}$ recognizes a^*.*

Let us now investigate the case where the set of matrices is commutative.

Let $L \in \Sigma^*$ be a language. The *commutative closure* of L is defined as $com(L) = \{x \in \Sigma^* | \phi(x) \in \phi(L)\}$. A language is *commutative* if $com(L) = L$.

Theorem 4.29. *If a language L is recognized by a* 0-XBHVA *$\mathcal{V}$ where $X \in$* {D,N} *with a commutative set of matrices, then L is commutative.*

Proof. Let $w \in L$ and suppose that the string $w = w_{[1]}w_{[2]}\cdots w_{[n]}$ is accepted by $\mathcal{V}$. Let $A_1A_2\cdots A_n$ be the product of the matrices labeling the computation such that

$$vA_1A_2\cdots A_n = v$$

where v is the initial vector of $\mathcal{V}$. Since the matrices are commutative, then for any permutation τ,

$$A_1A_2\cdots A_n = A_{\tau(1)}A_{\tau(2)}\cdots A_{\tau(n)}.$$

This leads to the acceptance of the string $w' = w_{[\tau(1)]}w_{[\tau(2)]}\cdots w_{[\tau(n)]}$ since

$$vA_{\tau(1)}A_{\tau(2)}\cdots A_{\tau(n)} = v.$$

Hence, if w is accepted by $\mathcal{V}$, then any string obtained from w by permuting its letters is also accepted by $\mathcal{V}$. Any string x with $\phi(x) = \phi(w)$ is in L and we conclude that L is commutative. □

When the computation is not blind, then the class of languages recognized is no longer commutative. The language of balanced strings of brackets `DYCK` can be recognized by 0-DHVA(1) as follows. Starting with the initial vector (1), for each left bracket the vector is multiplied by (2). As long as the vector is not equal to (1), for each right bracket, the vector is multiplied by $(\frac{1}{2})$. If the vector is equal to (1) and the next symbol is a right bracket, then the vector is set to (0).

Corollary 4.30. *If a language L is recognized by a* 0-XBHVA $\mathcal{V}$ *where* $X \in \{\text{D,N}\}$ *with a commutative set of matrices, then* $L = L^r$.

Proof. Suppose that $w \in L$. Then it is clear that w^r will be also accepted by $\mathcal{V}$ by Theorem 4.29 and $w^r \in L$. Since for every string w it is true that $w \in L$ if and only if $w^r \in L$, we conclude that $L = L^r$. □

4.6.4.2. Regular languages. Let $\mathcal{F}$ be an n-state deterministic finite automaton. Without loss of generality, we can enumerate its states as $q_1, \ldots, q_n$ where q_1 is the initial state. Moreover we can denote q_i by e_i, which is the basis vector in dimension n having 1 in its i'th entry, and 0 otherwise. Besides, for each symbol σ, we can design a zero-one matrix, say A_σ, that represents the transitions between the states, i.e. $A_\sigma[i, j] = 1$ if and only if $\mathcal{F}$ switches from q_i to q_j when reading symbol σ. Thus, the computation of $\mathcal{F}$ on an input, say w, can be traced by matrix-vector multiplication:

$$e_j = e_1 A_{w[1]} A_{w[2]} \cdots A_{w[|w|]}$$

if and only if $\mathcal{F}$ ends its computation in q_j after reading w.

Based on this representation, we can easily observe that if a language L is recognized by an n-DFA whose initial state is the single accept state, then L is recognized by a 0-DBHVA(n).

Let us give some examples of stateless HVAs recognizing regular languages.

Example 4.31. *For $k > 1$, $\mathtt{AB_k}^* = \{a^k b^k\}^* \in \mathfrak{L}$(0-DBHVA(2k)).*

Let us construct a 0-DBHVA($2k$) $\mathcal{V}_k$ recognizing $\mathtt{AB_k}^*$. The initial vector of $\mathcal{V}_k$ is $v = (1 \;\; 0 \;\; \cdots \;\; 0)$. For each a, the value in the i'th entry of the vector is transferred to the $(i+1)$'st entry when $1 \leq i \leq k$, and, the rest of the entries are set to 0. For each b, the value in the $(i+k)$'th entry of the vector is transferred to the $(i+k+1 \mod 2k)$'th entry, and the rest of the entries are set to 0, $(1 \leq i \leq k)$. Thus, we return to the initial vector if and only if after some number of $(a^k b^k)$ blocks have been read.

Example 4.32. $\mathtt{MOD_m} = \{a^i \mid i \mod m \equiv 0\} \in \mathfrak{L}$(0-DBHVA($m$)).

It is easy to observe that any unary n-state DFA whose initial state is the single accept state can recognize either L_ε or $\mathtt{MOD_m}$ for some $m \leq n$. Hence, for any $m > 0$, the language $\mathtt{MOD_m}$ is recognized by a 0-DBHVA(m).

Note that if it is allowed to use arbitrary algebraic numbers in the transition matrices, then for every $m > 0$, the language $\texttt{MOD}_\texttt{m}$ is recognized by a 0-DBHVA(2) with initial vector $v = (1 \;\; 0)$ that multiplies its vector with the matrix

$$A_m = \begin{pmatrix} \cos\frac{2\pi}{m} & -\sin\frac{2\pi}{m} \\ \sin\frac{2\pi}{m} & \cos\frac{2\pi}{m} \end{pmatrix}$$

for each scanned symbol. (The entries of A_m are algebraic for any m [75].)

One may ask whether all regular languages L satisfying $L = L^*$ can be recognized by stateless HVAs. We provide a negative answer for the deterministic model. The language MOD23 is defined as $\{a^2, a^3\}^* = a^* \setminus \{a\}$, satisfying that $\texttt{MOD23} = \texttt{MOD23}^*$. MOD23 cannot be recognized by any 0-DHVA since a 0-DHVA which accepts a^2 and a^3, also accepts a by Lemma 4.26.

4.6.4.3. Stateless 1-dimensional HVAs. We now focus on stateless HVAs whose vectors have dimension 1 and demonstrate some results on stateless FAMWs. Note that stateless FAMs do not process the end-marker \$ by definition, since their single state is also an accept state and the computation ends once \$ is scanned in an accept state.

We start by comparing the class of languages recognized by stateless versions of kBCAs, FAMWs, and BHVA(1)s. The equivalence between kBCA and FAMWs and BHVA(1)s does not hold immediately in the stateless case. The reason is that the counters can only be updated by the set $\{-1, 0, 1\}$ in a single step since additional states are needed to update the counters by arbitrary integers (see Fact 2.1). Furthermore, the capability of multiplication with negative rational numbers brings additional power to the stateless DBHVA(1)s.

Theorem 4.33. $\mathfrak{L}$(0-XFAMW) $\subsetneq$ $\mathfrak{L}$(0-XBHVA(1)) *where* $X \in \{\text{D,N}\}$.

Proof. Let $\texttt{EVENAB} = \{a^n b^n |\ n = 2k \text{ for some } k \geq 0\}$. The following 0-DBHVA(1) recognizes EVENAB: The register is multiplied by (-2) and $(1/2)$ when the machine

reads an a and b respectively.

Suppose that there exists some 0-XFAMW recognizing EVENAB. Let m_a and m_b be the positive rational numbers that are multiplied by the register upon reading a and b. Since $aabb \in$ EVENAB, it is true that $m_a^2 m_b^2 = 1$. Since both m_a and m_b are positive, it is not possible that $m_a m_b = -1$. It follows that $m_a m_b = 1$, in which case the string ab is accepted and we get a contradiction. Hence we conclude that EVENAB cannot be recognized by any 0-XFAMW. □

When the register is multiplied with only positive rational numbers, then we have $\mathfrak{L}$(0-XFAMW)=$\mathfrak{L}$(0-XBHVA(1))$_{\mathbb{Q}^+}$.

For real-time and blind machines, we can state the following result.

Corollary 4.34. *If $L \in \mathfrak{L}$(0-XFAMW) where $X \in$ {D,N}, then L is commutative.*

Proof. By Theorem 4.33, L is accepted by a 0-XBHVA(1). Since multiplication in dimension 1 is commutative, the result follows by Theorem 4.29. □

Let us recall Fact 2.3, which states that a bounded language is accepted by a 1NFAMW iff it is semilinear [6]. In the next theorem, we prove a similar result and characterize the class of languages recognized by 0-DFAMWs. We show that any language recognized by a 0-DFAMW is commutative and semilinear. Furthermore, any commutative language whose Parikh image is the set of nonnegative solutions to a system of linear homogenous Diophantine equations can be recognized by a 0-DFAMW.

Theorem 4.35. *$L \in \mathfrak{L}$(0-DFAMW) iff L is commutative and $\phi(L)$ is the set of nonnegative integer solutions to a system of linear homogeneous Diophantine equations.*

Proof. Let L be a language over the alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ recognized by a 0-DFAMW $\mathcal{V}$. Let $A = \{a_1, a_2, \ldots, a_n\}$ be the set of rational numbers such that the

register is multiplied with a_i upon reading σ_i. Let $\{p_1, p_2, \ldots, p_k\}$ be the set of prime factors of the denominators and the numerators of the rational numbers in A. Then each a_i can be expressed as

$$a_i = \frac{p_1^{x_{1_i}} p_2^{x_{2_i}} \cdots p_k^{x_{k_i}}}{p_1^{y_{1_i}} p_2^{y_{2_i}} \cdots p_k^{y_{k_i}}}.$$

If a string w is accepted by $\mathcal{V}$, then the value of the register should be equal to 1 after reading w, which is possible only if

$$a_1^{w_{|\sigma_1|}} a_2^{w_{|\sigma_2|}} \cdots a_n^{w_{|\sigma_n|}} = 1.$$

This leads to the following system of linear Diophantine equations in n variables.

$$\begin{aligned}
(x_{1_1} - y_{1_1})w_{|\sigma_1|} + (x_{1_2} - y_{1_2})w_{|\sigma_2|} + \cdots + (x_{1_n} - y_{1_n})w_{|\sigma_n|} &= 0 \\
(x_{2_1} - y_{2_1})w_{|\sigma_1|} + (x_{2_2} - y_{2_2})w_{|\sigma_2|} + \cdots + (x_{2_n} - y_{2_n})w_{|\sigma_n|} &= 0 \\
&\vdots \\
(x_{k_1} - y_{k_1})w_{|\sigma_1|} + (x_{k_2} - y_{k_2})w_{|\sigma_2|} + \cdots + (x_{k_n} - y_{k_n})w_{|\sigma_n|} &= 0
\end{aligned}$$

For $j = 1, \ldots, k$, the j'th equation is stating that the exponent of p_j is equal to 0 after reading w.

Hence, we can conclude that the Parikh images of the accepted strings are the nonnegative solutions to a system of linear homogeneous Diophantine equations. L is commutative by Theorem 4.34. (One can further conclude that L is semilinear.)

For the converse, suppose that we are given a commutative language L over the alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$. Let T be the set of Parikh images of the strings in L. T is the set of nonnegative solutions to a system of, say, k linear homogeneous Diophantine equations in n unknowns,

$$\begin{aligned} b_{11}t_1 + b_{12}t_2 + \cdots + b_{1n}t_n &= 0 \\ b_{21}t_2 + b_{22}t_2 + \cdots + b_{2n}t_n &= 0 \\ &\vdots \\ b_{k1}t_1 + b_{k2}t_2 + \cdots + b_{kn}t_n &= 0 \end{aligned}$$

where $(t_1 \quad t_2 \quad \ldots \quad t_n) \in T$.

We construct a 0-DFAMW $\mathcal{V}$ recognizing L as follows. We choose a set of k distinct prime numbers, $\{p_1, p_2, \ldots, p_k\}$. When $\mathcal{V}$ reads σ_i, the register is multiplied by

$$a_i = p_1^{b_{1i}} p_2^{b_{2i}} \cdots p_k^{b_{ki}}.$$

Suppose that a string w is accepted by $\mathcal{V}$. Then

$$a_1^{w_{|\sigma_1|}} a_2^{w_{|\sigma_2|}} \cdots a_n^{w_{|\sigma_n|}} = 1.$$

The product is equal to 1 if all of the exponents of the prime factors are equal to 0, that is when $\begin{pmatrix} w_{|\sigma_1|} & w_{|\sigma_2|} & \ldots & w_{|\sigma_n|} \end{pmatrix} \in T$. Hence we see that the set of accepted strings are those with Parikh image in T. Since L is commutative, any string w with $\phi(w) \in T$ belongs to L and we conclude that $\mathcal{V}$ recognizes L.

□

Note that $\mathfrak{L}$(0-DFAMW)=$\mathfrak{L}$(0-1DFAMW), since a 0-1DFAMW that has an instruction to stay on some input symbol cannot read the rest of the string. The reasoning of Lemma 4.8 applies.

4.6.4.4. Additional Results on Stateless Homing Vector Automata. One sees that the nonregular language $\texttt{AB} = \{a^n b^n | n \geq 0\}$ cannot be recognized by any 0-NHVA(k) for any k, since $\texttt{AB} \neq \texttt{AB}^*$. On the other hand, $\texttt{EQ} = \{x \in \{a,b\}^* \mid |x|_a = |x|_b\}$ can be recognized by a 0-DBHVA(1) with initial vector (1), and transition matrices $A_a = (2)$ and $A_b = (1/2)$. It is possible to recognize the star of AB, even in the deterministic and blind computation mode with a stateless homing vector automaton. Note that $\texttt{AB}^*$ cannot be recognized by any 1NFAMW [6]. The proof is due to Abuzer Yakaryılmaz and can be found in [17].

Theorem 4.36. *The language* $\texttt{AB}^* = \{\{a^n b^n\}^* \mid n \geq 0\}$ *is recognized by a* 0-DBHVA(10).

Nondeterministic HVAs are more powerful than their deterministic variants in terms of language recognition in general. In the next theorem, we show that this is also true for the stateless models.

Theorem 4.37. *(i)* $\mathfrak{L}$(0-DBHVA) $\subsetneq$ $\mathfrak{L}$(0-NBHVA).
(ii) $\mathfrak{L}$(0-DHVA) $\subsetneq$ $\mathfrak{L}$(0-NHVA).

Proof. Let us construct a 0-NBHVA(1) $\mathcal{V}$ recognizing $\texttt{LEQ} = \{x \in \{a,b\}^* \mid |x|_a \leq |x|_b\}$. Starting with the initial vector (1), $\mathcal{V}$ multiplies its vector with $A = (2)$ for each a and with $B_1 = (1/2)$ or $B_2 = (1)$ for each b nondeterministically.

Suppose that LEQ can be recognized by a 0-DHVA(k) $\mathcal{V}'$. The strings $w_1 = b$ and $w_2 = ba$ are accepted by $\mathcal{V}'$. By Lemma 4.26, the string $w_3 = a$ is also accepted by $\mathcal{V}'$. We obtain a contradiction, and conclude that LEQ cannot be recognized by any 0-DHVA(k). □

Now let us compare the language recognition power of 1 and 2 dimensional stateless HVAs.

Theorem 4.38. *(i)* $\mathfrak{L}(\text{0-DBHVA}(1)) \subsetneq \mathfrak{L}(\text{0-DBHVA}(2))$.
(ii) $\mathfrak{L}(\text{0-NBHVA}(1)) \subsetneq \mathfrak{L}(\text{0-NBHVA}(2))$.

Proof. Note that the language $\mathtt{AB_1}^* = (ab)^*$ can be recognized by a 0-DBHVA(2) by Example 4.32. Assume that $\mathtt{AB_1}^*$ is recognized by a 0-NBHVA(1). Since $\mathtt{AB_1}^*$ is not equal to its reverse, by Corollary 4.30 we get a contradiction. We conclude that $\mathtt{AB_1}^*$ cannot be recognized by any 0-NBHVA(1). □

Now, we compare the blind and non-blind versions of one dimensional HVAs.

Theorem 4.39. *(i)* $\mathfrak{L}(\text{0-DBHVA}(1)) \subsetneq \mathfrak{L}(\text{0-DHVA}(1))$.
(ii) $\mathfrak{L}(\text{0-NBHVA}(1)) \subsetneq \mathfrak{L}(\text{0-NHVA}(1))$.

Proof. Let us construct a 0-DHVA(1) $\mathcal{V}$ recognizing $\mathtt{AB_1}^* = (ab)^*$. The initial vector is equal to (1). For each scanned a, $\mathcal{V}$ multiplies its vector with $A_= = (2)$ if it is equal to its initial value, and with $A_{\neq} = (0)$ otherwise. For each scanned b, $\mathcal{V}$ multiplies its vector with $B_= = (0)$ if it is equal to its initial value, and with $B_{\neq} = (1/2)$ otherwise. $\mathtt{AB_1}^*$ cannot be recognized by any 0-NBHVA(1) as we saw in the proof of Theorem 4.38. □

Let us look at some closure properties for the stateless models. All of the classes associated with the stateless models are closed under the star operation since for any language recognized by a stateless homing vector automaton, it is true that $L = L^*$.

Theorem 4.40. *(i)* $\bigcup_k \mathfrak{L}(\text{0-DBHVA}(k))$ *is closed under the following operation:*
(a) intersection
(ii) $\bigcup_k \mathfrak{L}(\text{0-DBHVA}(k))$ *and* $\mathfrak{L}(\text{0-DHVA})$ *are not closed under the following operations:*
(a) union

(b) complement

(c) concatenation

Proof.

(i) (a) The proof of Theorem 4.25 that $\bigcup_k \mathfrak{L}(\text{DBHVA}(k))$ is closed under intersection is also valid for stateless models.

(ii) (a) The languages $\texttt{MOD}_2$ and $\texttt{MOD}_3$ are recognized by 0-DBHVA(2) and 0-DBHVA(3) respectively, by Example 4.32. Their union cannot be recognized by any 0-DHVA(k), since a 0-DHVA(k) accepting the strings a^2 and a^3 should also accept the non-member string a by Lemma 4.26.

(b) The complement of the language $\texttt{MOD}_\texttt{m}$ ($m > 1$)is not recognized by any 0-NHVA(k), since $\overline{\texttt{MOD}_\texttt{m}}$ contains a and any 0-NHVA(k) accepting a accepts any member of a^*.

(c) The concatenation of $\texttt{MOD}_2$ and $\texttt{MOD}_3$, MOD23, cannot be recognized by any 0-DHVA.

□

$\bigcup_k \mathfrak{L}$(0-NBHVA(k)) and $\bigcup_k \mathfrak{L}$(0-NHVA(k)) are not closed under complement. $\bigcup_k \mathfrak{L}$(0-NBHVA(k)) is closed under intersection. The proofs are identical.

4.7. Open Questions

What can we say about the relationship between real-time homing vector automata and one-way homing vector automata? We conjecture that one-way nondeterministic blind homing vector automata are more powerful than their real-time versions. Our candidate language is $\texttt{UPOW} = \{a^{2^n} | n \geq 0\}$, which can be recognized by a 1NBHVA(2). Note that when the machine in consideration is deterministic and blind, the real-time and one-way versions are equivalent in power.

Can we show a separation result between the class of languages recognized based on the set of matrices used during the transitions of a homing vector automaton?

Can we show a hierarchy result between the classes of languages recognized by deterministic homing vector automata of dimensions k and $k+1$ for some $k > 1$, maybe when the matrix entries are restricted to the set $\{-1, 0, 1\}$? Consider the family of languages $\texttt{POW}(k) = \{a^n b^{k^n} | n \geq 0\}$. We conjecture that it is not possible to recognize $\texttt{POW}(k)$ with a homing vector automaton of dimension less than $k+1$ with the restricted set of matrices.

Let G be a group of $k \times k$ matrices. Can we always construct a 1NBHVA$(k)_G$ recognizing the same language as a given G-automaton? (Note that we have proven that this is the case for 1NBHVA$(2)_{\mathbf{F}_2}$ and $\mathbf{F}_2$-automaton.)

Suppose that one can always find a suitable initial vector v such that for every $A \in G$ except the identity matrix, $vA \neq v$. Then one could construct the required 1NBHVA$(k)_G$ from the given G-automaton directly. For which groups G is it always possible to find such a vector?

What can we say about the reverse direction? For instance, is every language recognized by some 1NBHVA$(2)_{\mathbf{F}_2}$ context-free?

We proved that 1NBHVA(k)s are more powerful than extended finite automata when both are defined over 2×2 integer matrices. Is this result still true when both models are defined over 3×3 integer matrices?

Do 0-NHVA(k)s recognize every regular language L satisfying $L = L^*$? Is there any nonregular language L satisfying $L = L^*$ that cannot be recognized by any stateless HVA?

We proved that any language recognized by a 0-NFAMW is commutative. What can we say about the non-blind case?

We gave a characterization for the class of languages recognized by 0-DFAMWs. Can we give a similar characterization for the non-blind and nondeterministic models?

5. VALENCE GRAMMARS AND VALENCE AUTOMATA

In this chapter, we are going to focus on context-free valence grammars, valence automata and valence pushdown automata.

We start by stating the formal definitions in Section 5.1. In Section 5.2, we show that valence pushdown automata and finite valence automata are equivalent in terms of language recognition power, using the well known equivalence between pushdown automata and finite automata over polycyclic monoids. Then, we extend some results proven for valence automata in [35] to context-free valence grammars and give the properties of the set of languages generated by context-free valence grammars in Section 5.3. We state some open questions in Section 5.4.

5.1. Definitions

Let $\mathcal{G} = (N, T, P, S)$ be a *context-free grammar* where N is the set of variables, T is the terminal alphabet, $P \subseteq N \times (N \cup T)^*$ is the set of rules or productions, and $S \in N$ is the start symbol. We will denote by $\Rightarrow$ and $\Rightarrow^*$ the step derivation relation and its regular closure respectively. $L(\mathcal{G})$ denotes the language $\{w \in T^* : S \Rightarrow^* w\}$ of words generated by G.

Given a monoid M, a *context-free valence grammar over* M [8] is a five-tuple $\mathfrak{G} = (N, T, P, S, M)$, where N, T, S are defined as before and $P \subseteq N \times (N \cup T)^* \times M$ is a finite set of objects called *valence rules*. Every valence rule can be thus described as an ordered pair $p = (A \to \alpha, m)$, where $(A \to \alpha) \in N \times (N \cup T)^*$ and $m \in M$. The element m is called the valence of p.

The step derivation $(\Rightarrow)$ of the valence grammar is defined as follows: $(w, m) \Rightarrow (w', m')$ if there exists a valence rule $(A \to \alpha, n)$ such that $w = w_1 A w_2$ and $w' = w_1 \alpha w_2$ and $m' = mn$. The regular closure of $\Rightarrow$ will be denoted by $\Rightarrow^*$. A derivation of $\mathfrak{G}$ will be said *successful* or *valid* if it is of the form $(S, 1) \Rightarrow^* (w, 1)$, that is, it transforms the

pair $(S, 1)$ into the pair $(w, 1)$, after finitely many applications of the step derivation relation. The language generated by $\mathfrak{G}$ is the set of all the words w of T^* such that $(S, 1) \Rightarrow^* (w, 1)$.

A context-free valence grammar is said to be *regular* if all of its rules are right-linear, that is, every valence rule $(A \to \alpha, n)$ is such that $\alpha = uX$, where $u \in T^*$ and $X \in N$. The language families generated by context-free and regular valence grammars over M are denoted by $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M)$ and $\mathfrak{L}(\mathrm{Val}, \mathrm{REG}, M)$, respectively.

Let $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_1, Q_a)$ be a one-way nondeterministic pushdown automaton. Given a monoid M, the *nondeterministic valence pushdown automaton* (*valence* PDA) over M is the model of computation obtained from $\mathcal{A}$ as follows: with every transition of $\mathcal{A}$ is assigned an element of M, called the *valence* of the transition. Then the valence of an arbitrary computation is defined as the product of the valences of all the transitions of the computation (taken in the obvious order). A word of Σ^* is said to be *accepted* by the model if there exists an accepting computation for the word whose valence is the identity of M. The set of all the accepted words is defined as the language accepted by the valence pushdown automaton. The family of languages accepted by valence PDA over M is denoted $\mathfrak{L}(\mathrm{Val}, \mathrm{PDA}, M)$. It is worth noticing that the equivalence between valence pushdown automata and valence context-free grammars does not hold for an arbitrary monoid. However a result of [8] shows that the equivalence is true if M is a commutative monoid.

In the case that pushdown automaton $\mathcal{A}$ is a finite state automaton, the corresponding valence model is called the *valence automaton over* M. This model coincides with the M-automaton, and the family $\mathfrak{L}(\mathrm{Val}, \mathrm{NFA}, M)$ of languages accepted by valence automata is exactly $\mathfrak{L}(M)$. The equivalence between valence automata and regular valence grammars is proven in [8].

Introduced by [7], the valence grammars have been studied by various authors including [8, 76, 77]. A through study of several remarkable structural properties of the languages generated by the corresponding valence grammars has been done in [8], over

arbitrary monoids and in particular over commutative groups. In [8], the following fact is proven.

Fact 5.1. [8] *Context-free valence grammars over finite or commutative monoids are not stronger than context-free valence grammars over finite or commutative groups.*

It is left open in [8] whether context-free grammars with valences from finite monoids are more powerful than ordinary context-free grammars and the question is answered in [78].

Fact 5.2. [78] *Context-free grammars over finite monoids recognize exactly the class of context-free languages.*

Now we are going to present some definitions about monoids and semigroups.

Let S be a semigroup. Given subsets A and B of semigroups S, AB is the set $\{ab|a \in A, b \in B\}$. An *ideal* $\mathtt{I}$ of a semigroup S is a subset of S with the property that $S\mathtt{I}S \subseteq \mathtt{I}$.

The binary relation $\rho_{\mathtt{I}}$ defined by

$$a\rho_{\mathtt{I}}b \iff \text{either } a = b \text{ or both } a \text{ and } b \text{ belong to } \mathtt{I}$$

is a congruence. The equivalence classes of S mod $\rho_{\mathtt{I}}$ are $\mathtt{I}$ itself and every one-element set $\{x\}$ with $x \in S \setminus \mathtt{I}$. The quotient semigroup $S/\rho_{\mathtt{I}}$ is written as $S/\mathtt{I}$ and is called the *Rees quotient semigroup* [79].

$$S/\mathtt{I} = \{\mathtt{I}\} \cup \{\{x\}|x \in S \setminus \mathtt{I}\}$$

A semigroup is called *simple* if it contains no proper ideal. A semigroup S with a zero element is called 0-*simple* if the only ideals of S are $\{0\}$ and S itself, and $SS \neq \{0\}$.

5.2. Equivalence of Finite and Pushdown Automata with Valences

In this section, we are going to prove that valence pushdown automata are only as powerful as valence automata.

Let us recall the definition of polycyclic monoid which we have introduced in Subsection 3.3.3. Let X be a finite alphabet and let X^* be the free monoid of words over X. For each symbol $x \in X$, let P_x and Q_x be functions from X^* into X^* defined as follows: for every $u \in X^*$,

$$P_x(u) = ux, \quad Q_x(ux) = u.$$

The monoid of all partial functions on X^* generated by the set of functions $\{P_x,\, Q_x \,|x \in X\}$ is called the *polycyclic monoid* on X. We will focus on the polycyclic monoid of rank 2, which will be denoted by P_2, since it contains every polycyclic monoid of countable rank.

Theorem 5.3. *For any monoid M, $\mathfrak{L}(\mathrm{Val}, \mathrm{PDA}, M) = \mathfrak{L}(\mathrm{Val}, \mathrm{NFA},\ \mathrm{P}_2 \times M)$.*

Proof. Let $L \in \mathfrak{L}(\mathrm{Val}, \mathrm{PDA}, M)$ and $\mathfrak{P} = \{Q, \Sigma, X, \delta, q_1, Q_a, M\}$ be a valence PDA recognizing L. We know that a PDA with stack alphabet X is equivalent to a valence automaton over $P(X)$. Hence, $\mathfrak{P}$ can be seen as an NFA where two distinct valences (one in $P(X)$ and one in M) are assigned to each transition. An equivalent valence automaton $\mathcal{E} = \{Q, \Sigma, P(X) \times M, \delta', q_1, Q_a\}$ can be constructed, where a valence from the monoid $P(X) \times M$ is assigned to each transition. Recall that the partial functions Q_a and P_b model the operations of popping a and pushing b respectively. A transition of $\mathfrak{P}$ of the form $(q', b, m) \in \delta(q, \sigma, a)$ where $a, b \in X_\varepsilon$, $q, q' \in Q$, $\sigma \in \Sigma_\varepsilon$ and $m \in M$ can be expressed equivalently as $(q', (Q_a P_b, m)) \in \delta'(q, \sigma)$ where $(Q_a P_b, m) \in P(X) \times M$.

A string is accepted by $\mathcal{E}$ if and only if the product of the valences labeling the transitions in $\mathcal{E}$ is equal to $(1, 1)$, equivalently when the product of the valences labeling the transitions in $\mathfrak{P}$ is equal to the identity element of M and the stack is empty. Since

any polycyclic monoid is embedded in P_2, we conclude that $L \in \mathfrak{L}(\mathrm{Val}, \mathrm{NFA}, \mathrm{P}_2 \times M)$.

Conversely, let $L \in \mathfrak{L}(\mathrm{Val}, \mathrm{NFA}, \mathrm{P}_2 \times M)$ and let $\mathcal{E} = \{Q, \Sigma, \mathrm{P}_2 \times M, \delta, q_1, Q_a\}$ be a valence automaton over $\mathrm{P}_2 \times M$ recognizing L. Suppose that $(p, m) \in \mathrm{P}_2 \times M$ is a valence labeling a transition of $\mathcal{E}$. The product of the labels of a computation which involves a transition labeled by the zero element of P_2 cannot be equal to the identity element. Hence we can remove such transitions. Any nonzero element p of P_2 can be written as

$$Q_{x_1} Q_{x_2} \dots Q_{x_n} P_{y_1} P_{y_2} \dots P_{y_o}$$

for some $n, o \in \mathbb{N}$ and $x_i, y_i \in X_\varepsilon$, after canceling out elements of the form $P_a Q_a$ and $P_b Q_b$, where $X = \{a, b\}$ is the generator set for P_2. The product can be interpreted as a series of pop operations followed by a series of push operations performed by a PDA, without consuming any input symbol. Hence, an equivalent valence PDA $\mathfrak{P} = \{Q', \Sigma, X, \delta', q_1, Q_a, M\}$ can be constructed where a valence from M is assigned to each transition. Let $(q', (p, m)) \in \delta(q, \sigma)$ where $q, q' \in Q$, $\sigma \in \Sigma_\varepsilon$, $(p, m) \in \mathrm{P}_2 \times M$ and $p = Q_{x_1} Q_{x_2} \dots Q_{x_n} P_{y_1} P_{y_2} \dots P_{y_o}$ be a transition in $\mathcal{E}$. In $\mathfrak{P}$, we need additional $n + o$ states $\{q_1, \dots, q_{n+o}\} \notin Q$ and the following transitions to mimic that specific transition of $\mathcal{E}$.

$$\begin{aligned}
(q_1, \varepsilon, m) &\in \delta'(q, \sigma, x_1) \\
(q_2, \varepsilon, 1) &\in \delta'(q_1, \varepsilon, x_2) \\
&\vdots \\
(q_{n+1}, \varepsilon, 1) &\in \delta'(q_n, \varepsilon, x_n) \\
(q_{n+2}, y_1, 1) &\in \delta'(q_{n+1}, \varepsilon, \varepsilon) \\
(q_{n+3}, y_2, 1) &\in \delta'(q_{n+2}, \varepsilon, \varepsilon) \\
&\vdots \\
(q', y_o, 1) &\in \delta'(q_{n+o}, \varepsilon, \varepsilon)
\end{aligned}$$

A string is accepted by $\mathfrak{P}$ if and only if the product of the valences labeling the transitions in $\mathfrak{P}$ is equal to the identity element of M and the stack is empty, equivalently when the product of the valences labeling the transitions in $\mathcal{E}$ is equal to $(1, 1)$. We conclude that $L \in \mathfrak{L}(\mathrm{Val}, \mathrm{PDA}, M)$. □

Note that when M is commutative, the equality $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M) = \mathfrak{L}(\mathrm{Val}, \mathrm{NFA}, \mathrm{P}_2 \times M)$ also holds.

Corollary 5.4. *Let M be a polycyclic monoid of rank 2 or more. Then $\mathfrak{L}(\mathrm{Val}, \mathrm{PDA}, M)$ is the class of recursively enumerable languages.*

Proof. It is known that $\mathfrak{L}(\mathrm{Val}, \mathrm{NFA}, M \times M)$ is the class of recursively enumerable languages [33] when M is a polycyclic monoid of rank 2 or more. Since $\mathfrak{L}(\mathrm{Val}, \mathrm{PDA}, M) = \mathfrak{L}(\mathrm{Val}, \mathrm{NFA}, \mathrm{P}_2 \times M)$, by Theorem 5.3, the result follows. □

5.3. Context-free Valence Languages

It is known that the class of languages generated by regular valence grammars and the class of languages recognized by valence automata coincide [8]. In this section, we are going to prove that the results proven in [35] which hold for valence automata and therefore regular valence grammars, also hold for context-free valence grammars. Although the proofs are almost identical, they are presented here for completeness. Note that the same proofs can be also adapted to valence PDA.

In [35] Proposition 4.1.1, it is shown that the elements belonging to a proper ideal of a monoid do not have any use in the corresponding monoid automaton. We show that the same result holds for context-free valence grammars.

Proposition 5.5. *Let $\mathtt{I}$ be a proper ideal of a monoid M. Then $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M) = \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M/\mathtt{I})$.*

Proof. Let $L \in \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M)$ and let $\mathfrak{G}$ be a context-free valence grammar over the monoid M such that $L(\mathfrak{G}) = L$. The product of the valences which appear in a derivation containing a rule with valence $x \in \mathtt{I}$, will itself belong to $\mathtt{I}$. Since $\mathtt{I}$ is a proper ideal and $1 \notin \mathtt{I}$, such a derivation is not valid. Hence any such rules can be removed from the grammar and we can assume that $\mathfrak{G}$ has no such rules. For any $x_1, x_2, \ldots, x_n \in M \setminus \mathtt{I}$, it follows that $x_1 \ldots x_n = 1$ in M if and only if $\{x_1\}\{x_2\} \ldots \{x_n\} = \{1\}$ in $M/\mathtt{I}$. Let $\mathfrak{G}'$ be the context-free grammar with valences in $M/\mathtt{I}$, obtained from $\mathfrak{G}$ by replacing each valence $x \in M$ with $\{x\}$. It follows that a string w has a valid derivation in $\mathfrak{G}$ if and only if the product of the valences is mapped to $\{1\}$ in $\mathfrak{G}'$. Hence $L(\mathfrak{G}') = L$.

Conversely let $L \in \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M/\mathtt{I})$ and let $\mathfrak{G}'$ be a context-free grammar over the monoid $M/\mathtt{I}$ such that $L(\mathfrak{G}') = L$. Suppose that there exists a valid derivation consisting of a rule with $\mathtt{I}$ as the valence. Then the product of the valences of the whole derivation will be $\mathtt{I}$, which is not possible. Let $\mathfrak{G}$ be the context-free grammar with valences in M, obtained from $\mathfrak{G}'$ by replacing each valence $\{x\} \in M/\mathtt{I}$ with x. Since $\{x_1\}\{x_2\} \ldots \{x_n\} = \{1\}$ in $M/\mathtt{I}$ if and only if $x_1 \ldots x_n = 1$ in M, a string w has a valid derivation in $\mathfrak{G}$ if and only if the product of the valences is mapped to $\{1\}$ in $\mathfrak{G}'$. Hence $L(\mathfrak{G}) = L$. □

Let S be a semigroup. S is the *null semigroup* if it has an absorbing element zero and if the product of any two elements in S is equal to zero. A null semigroup with two elements is denoted by O_2.

The following corollary is analogous to [35] Cor. 4.1.2.

Corollary 5.6. *For every monoid M, there is a simple or 0-simple monoid N such that $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M) = \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, N)$.*

Proof. If M has no proper ideals then it is simple. Otherwise, let $\mathtt{I}$ be the union of all proper ideals of M and let $N = M/\mathtt{I}$. We can conclude from the proof of Cor. 4.1.2 [35] that $N^2 = 0$ or N is 0-simple. If $N^2 = 0$, then N is O_2 and the semigroup O_2 does not add any power to the grammar since it does not even contain

the identity element. Hence, $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, \mathrm{O}_2) = \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, \{1\})$ where $\{1\}$ is the trivial monoid which is simple. In the latter case N is 0-simple and by Proposition 5.5, $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M) = \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M/I) = \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, N)$. □

Proposition 4.1.3 of [35] states that a finite automaton over a monoid with a zero element is no more powerful then a finite automaton over a version of the same monoid from which the zero element has been removed, in terms of language recognition. The result is still true for context-free valence grammars since the same proof idea applies. The following notation is used: $M^0 = M \cup \{0\}$ if M has no zero element and $M^0 = M$ otherwise.

Proposition 5.7. *Let M be a monoid. Then $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M^0) = \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M)$.*

Proof. Since $M \subseteq M^0$, it follows that $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M) \subseteq \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M^0)$. Suppose $L \in \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M^0)$ and let $\mathfrak{G}$ be a context-free valence grammar with valences in M^0 and $L(\mathfrak{G}) = L$. Note that a valid derivation cannot contain a rule with a zero valence since otherwise the product of the valences would be equal to zero. Any such rules can be removed from $\mathfrak{G}$ to obtain $\mathfrak{G}'$, a context-free grammar with valences in M, without changing the language, and $L \in L(\mathfrak{G}')$. □

In the case $|X| = 1$, the monoid $P(X)$ coincides with the well-known structure of *bicyclic monoid* which will be denoted by B.

Fact 5.8. [35] *A simple (0-simple) monoid with identity 1 is either a group (respectively, a group with 0 adjoined) or contains a copy of the bicyclic monoid as a submonoid having 1 as its identity element.*

Theorem 5.9. *Let M be a monoid. Then either $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M) = \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, G)$ for some group G, or $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, N) \subseteq \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, \mathrm{B})$.*

Proof. Let M be a monoid. By Corollary 5.6, then $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M) = \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, N)$ for some simple or 0-simple monoid. By Fact 5.8, N is either a group (or a group

with zero adjoined) or contains a copy of the bicyclic monoid. If N is a group, then the result follows. If N is a group with zero adjoined, then by 5.7, $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, N) = \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, G)$ for some group G. Otherwise, it should be the case that $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, N) \subseteq \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, \mathrm{B})$. □

Now we are ready to prove the main theorem of the section which will allow us to determine the properties of the set of languages generated by context-free valence grammars. We need the following proposition which is the grammar analogue of Proposition 1 of [37].

Proposition 5.10. *Let M be a monoid, and suppose that L is accepted by a context-free valence grammar over M. Then there exists a finitely generated submonoid N of M such that L is accepted by a context-free valence grammar over N.*

Proof. There are only finitely many valences appearing in the rules of a grammar since the set of rules of a grammar is finite. Hence, the valences appearing in derivations are from the submonoid N of M generated by those elements. So the grammar can be viewed as a context-free valence grammar over N. □

Recall that a group G is *locally finite* if every finitely generated subgroup of G is finite and *periodic* if every element of the group has finite order.

Theorem 5.11. *Let M be a monoid. Then $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M)$ either*

(i) equals CF,
(ii) contains $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, \mathbb{Z})$,
(iii) contains $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, \mathrm{B})$ *or*
(iv) is equal to $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, G)$ *for G an infinite periodic group which is not locally finite.*

Proof. Let M be a monoid. Then either $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M) = \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, G)$ for some group G, or $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, N) \subseteq \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, \mathrm{B})$. Then in the former case *(iii)* holds. In the latter case $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M) = \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, G)$ for some group N.

In the latter case, if N is a group with zero adjoined, then by Proposition 5.7 we know that $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, N) = \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, G)$ for some group G. If G is not periodic, then it has an element of infinite order which generates a subgroup isomorphic to $\mathbb{Z}$ and hence *(iii)* follows. Otherwise, suppose that G is locally finite. By Proposition 5.10, every language in $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, G)$ belongs to $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, H)$ for some finitely generated subgroup H of G. Since G is locally finite, H is finite. Any language $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, H)$ is context-free by a result from [78] and hence (i) holds. The only remaining case is that G is a periodic group which is not locally finite, in which case (iv) holds. □

The result about valence grammars over commutative monoids which we have stated in Fact 5.1, now follows as a corollary of Theorem 5.11.

Corollary 5.12. *Let M be a commutative monoid. Then $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M) = \mathfrak{L}(\mathrm{Val}, \mathrm{CF}, G)$ for some group G.*

Proof. Since no commutative monoid M can contain a copy of the bicyclic monoid as a submonoid, the result follows by the proof of Theorem 5.11. □

5.4. Open Questions

We proved that a valence PDA over M is equivalent to a valence automaton over $\mathrm{P}_2 \times M$. Can we prove a similar equivalence result for context-free valence grammars?

In Theorem 5.11, we conclude that $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, M)$ contains the class $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, \mathrm{B})$ when M is a monoid that contains B. Since B is not commutative, no correspondence with valence PDA exists, and little is known about the class $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, \mathrm{B})$, except that it contains the set of partially blind one counter languages. What can we say further about $\mathfrak{L}(\mathrm{Val}, \mathrm{CF}, \mathrm{B})$?

6. CONCLUSION

The main purpose of this thesis is to explore various computational models with storage. We mainly examine extended finite automata and homing vector automata, investigating the language recognition power of these models under different restrictions. We also present several results about valence pushdown automata and context-free valence grammars.

Matrices have a fundamental role in this study. Focusing on finite automata over matrix groups, both models can be regarded as a finite automaton equipped with a storage mechanism that is modified through matrix multiplications.

We first examine the class of languages recognized by finite automata over rational and integer matrix groups and compare them with the previously known language classes. Our findings can be summarized as follows:

- Finite automata over the group of 2×2 integer matrices recognize exactly the class of context-free languages.
- Finite automata over the group of 2×2 rational matrices with determinant 1 can recognize some non-context-free languages.

Together with the previous results, the overall picture is given in Figure 6.1. The question mark indicates that the existence of a language in the particular subset is unknown.

Next we analyze the language recognition power of extended finite automata under time restriction. We prove that the growth rate of the group is effective in the language recognition power of the corresponding group automaton, when a time bound is imposed on the machine. Using this result we prove that there exists a context-free language which cannot be recognized by any G-automata in polynomial time if the group G has polynomial growth. Furthermore, we investigate the class of languages

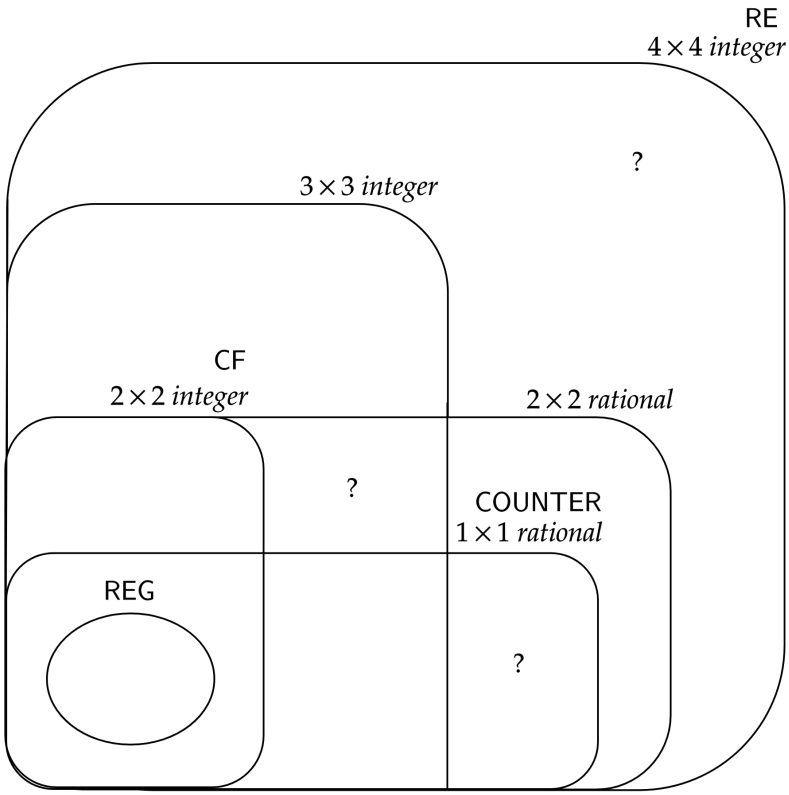


Figure 6.1. The hierarchy of the classes of languages recognized by extended finite automata over integer and rational matrix groups

recognized by group automata in linear-time and obtain the following results:

- Free group automata recognize exactly the class of context-free languages in linear time.
- Heisenberg group automata cannot recognize some context-free languages in linear time.
- The class of languages recognized by finite automata over the group of 3×3 integer matrices is a proper superclass of the class of languages recognized by Heisenberg group automata under the restriction of linear time.

We investigate the link between the decidability problems for matrix groups and the corresponding group automata. We provide alternative proofs for the decidability of the subsemigroup membership problem for the group of 2×2 integer matrices and

the identity problem for the monoid of 2×2 matrices with integer entries. We also prove the undecidability of the emptiness problem for finite automata over 4×4 integer matrix groups.

Another way we focus on matrices is through the study of the homing vector automaton model, a finite automaton equipped with a vector. Like in many classical models where the acceptance condition is ending with the initial register value, a string is accepted by a homing vector automaton if the vector is equal to its initial value after a series of multiplications with some rational valued matrices. We define different variants of the machine such as real-time, one-way, deterministic, nondeterministic, blind, and non-blind and we add further restrictions on the machine by restricting the matrices multiplied with the register to a specific set. We investigate the relationship between homing vector automata and counter automata, proving that one-dimensional homing vector automata are equivalent to blind counter automata and the two models are incomparable when the computation is not blind. One-way nondeterministic blind homing vector automata are closely linked to extended finite automata over matrix groups and this link extends our knowledge on homing vector automata. We visualize our findings in Figure 6.2 and summarize our findings as follows:

- The class of languages recognized by one-way nondeterministic blind homing vector automata over the group of 2×2 integer matrices with determinant 1 contains the class of context-free languages.
- Over the monoid of 2×2 integer matrices, the class of languages recognized by one-way nondeterministic blind homing vector automata is a proper superset of the class of languages recognized by extended finite automata.
- Over the monoid of 3×3 integer matrices, the class of languages recognized by one-way nondeterministic blind homing vector automata is a proper superset of the class of languages recognized by one-way nondeterministic blind counter automata.
- The class of languages recognized by one-way nondeterministic blind homing vector automata over the group of 4×4 integer matrices with determinant 1 is

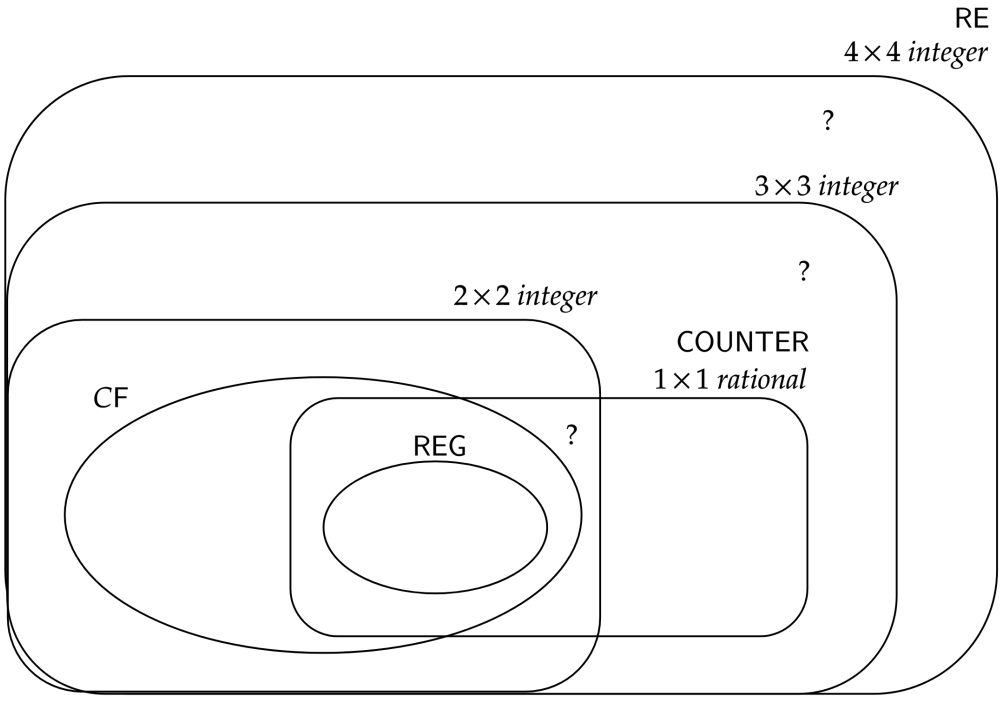


Figure 6.2. The hierarchy of languages recognized by one-way nondeterministic blind homing vector automata with rational and integer entries

the class of recursively enumerable languages.

We propose a new method called the generalized Stern-Brocot encoding, which enables encoding any string over an alphabet of size k into a k-dimensional vector. The encoding can be carried out in real-time with $k \times k$ matrices whose entries belong to the set $\{-1, 0, 1\}$. We further explore real-time homing vector automata, by establishing some separation results among the various variants and discussing some closure properties. We prove that any unary language recognized by a real-time deterministic homing vector automaton is regular. Continuing our study on real-time homing vector automata, we investigate their stateless versions. We examine the language recognition power of different versions of these machines and make some observations. Our study of the stateless real-time deterministic blind one-dimensional homing vector automata yields the following characterization for the class of languages recognized by stateless deterministic finite automata with multiplication without equality.

- A language is recognized by a stateless deterministic finite automata with multiplication without equality iff it is commutative and its Parikh image is the set of nonnegative integer solutions to a system of linear homogeneous Diophantine equations.

Finally we look at the problem of string separation by homing vector automata and vector automata, observing that these models can separate any pair of words by using 2-dimensional vectors in the real-time, deterministic and blind computation mode.

Other computational models we investigate are the valence pushdown automata and context-free valence grammars. We prove that valence pushdown automata are equivalent to valence automata defined over some specific monoid. We generalize some results proven in the context of extended finite automata to context-free valence grammars.

To conclude, this research opens up a new perspective for analyzing the decision problems of matrix groups and various automaton models, and any computational model than can be traced by matrix multiplications. The study of extended finite automata over matrix groups and under time restriction contributes to the current literature on the topic. The classes of languages recognized by extended finite automata and homing vector automata over integer matrices of dimensions 2, 3, and 4 provides a different point of view for classifying languages.

Many questions in need of further investigation are presented throughout the thesis. The future research on the subject should concentrate on the language recognition power of extended finite automata and homing vector automata over 3×3 integer matrices. We conjecture that these classes are the proper subsets of the class of recursively enumerable languages. Note that for the group of 3×3 integer matrices, the decidability of the membership problem is also open.

REFERENCES

1. Sipser, M., *Introduction to the Theory of Computation, 2nd edition*, Thomson Course Technology, United States of America, 2006.

2. Chomsky, N., "Context-free grammars and pushdown storage", *M. I. T. Res. Lab. Electron. Quart. Prog. Report.*, Vol. 65, pp. 187–194, 1962.

3. Fischer, P. C., A. R. Meyer and A. L. Rosenberg, "Real time counter machines", *Proceedings of the 8th Annual Symposium on Switching and Automata Theory, SWAT'67 (FOCS)*, pp. 148–154, 1967.

4. Turing, A. M., "On computable numbers, with an application to the Entscheidungsproblem", *Proceedings of the London mathematical society*, Vol. 2, No. 1, pp. 230–265, 1937.

5. Mitrana, V. and R. Stiebe, "The accepting power of finite automata over groups", *New Trends in Formal Languages*, pp. 39–48, Springer-Verlag, 1997.

6. Ibarra, O. H., S. K. Sahni and C. E. Kim, "Finite automata with multiplication", *Theoretical Computer Science*, Vol. 2, No. 3, pp. 271 – 294, 1976.

7. Păun, G., "A new generative device: valence grammars", *Rev. Roumaine Math. Pures Appl*, Vol. 25, No. 6, pp. 911–924, 1980.

8. Fernau, H. and R. Stiebe, "Sequential grammars and automata with valences", *Theoretical Computer Science*, Vol. 276, No. 1–2, pp. 377 – 405, 2002.

9. Salehi, Ö., A. Yakaryılmaz and A. C. C. Say, "Real-time vector automata", *Proceedings of the 19th International Conference on Fundamentals of Computation Theory, FCT'13*, Vol. 8070 of *LNCS*, pp. 293–304, Springer, 2013.

10. Salehi, Ö., F. D'Alessandro and A. C. C. Say, "Language classes associated with automata over matrix groups", *Proceedings of the Eighth Workshop on Non-Classical Models of Automata and Applications, NCMA'16*, Vol. 321 of *books@ocg.at*, pp. 287–300, Österreichische Computer Gesellschaft, 2016.

11. Salehi, Ö., F. D'Alessandro and A. C. Say, "Language classes associated with automata over matrix groups", *RAIRO-Theoretical Informatics and Applications*, Vol. 52, No. 2-3-4, pp. 253–268, 2018.

12. Salehi, Ö. and A. C. C. Say, "Extended finite automata and decision problems for matrix semigroups.", *Tenth Workshop on Non-Classical Models of Automata and Applications (Short Papers)*, Vol. 321, pp. 45–52, Österreichische Computer Gesellschaft, 2018.

13. Turakainen, P., "Generalized automata and stochastic languages", *Proceedings of the American Mathematical Society*, Vol. 21, pp. 303–309, 1969.

14. Lipton, R. J. and K. W. Regan, *Quantum Algorithms via Linear Algebra*, MIT Press, 2014.

15. Salehi, Ö. and A. C. C. Say, "Homing vector automata", *Proceedings of the Seventh Workshop on Non-Classical Models of Automata and Applications, NCMA'15*, Vol. 318 of *books@ocg.at*, pp. 193–205, Österreichische Computer Gesellschaft, 2015.

16. Salehi, Ö., A. C. C. Say and F. D'Alessandro, "Homing vector automata", *RAIRO - Theoretical Informatics and Applications*, Vol. 50, No. 4, pp. 371–386, 2016.

17. Salehi, Ö., A. Yakaryılmaz and A. C. C. Say, "New results on vector automata and homing vector automata", *International Journal of Foundations of Computer Science*, 2019, accepted.

18. Salehi, Ö., F. D'Alessandro and A. C. C. Say, "Generalized results on monoids as memory", *Proceedings of the 15th International Conference on Automata and*

Formal Languages,AFL'17, Vol. 252 of *EPTCS*, pp. 234–247, 2017.

19. Minsky, M., *Recursive Unsolvability of Post's Problem of "tag": And Other Topics in Theory of Truing Machines*, Massachusetts Institute of Technology, Lincoln Laboratory, 1960.

20. Fischer, P. C., A. R. Meyer and A. L. Rosenberg, "Counter machines and counter languages", *Mathematical Systems Theory*, Vol. 2, No. 3, pp. 265–283, 1968.

21. Greibach, S. A., "Remarks on the complexity of nondeterministic counter languages", *Theoretical Computer Science*, Vol. 1, No. 4, pp. 269 – 288, 1976.

22. Petersen, H., "Simulations by time-bounded counter machines", *International Journal of Foundations of Computer Science*, Vol. 22, pp. 395–409, 2011.

23. Greibach, S. A., "Remarks on blind and partially blind one-way multicounter machines", *Theoretical Computer Science*, Vol. 7, pp. 311–324, 1978.

24. Fraleigh, J. and V. Katz, *A First Course in Abstract Algebra*, Addison-Wesley world student series, Addison-Wesley, 2003.

25. Lyndon, R. C. and P. E. Schupp, *Combinatorial Group Theory*, Springer-Verlag, 1977.

26. Schreier, O., "Die Untergruppen der freien Gruppen", *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, Vol. 5, No. 1, pp. 161–183, 1927.

27. Nielsen, J., "Om Regning med ikke-kommutative Faktorer og dens Anvendelse i Gruppeteorien", *Matematisk Tidsskrift. B*, pp. 77–94, 1921.

28. Gilman, R., "Formal languages and infinite groups", *Geometric and computational perspectives on infinite groups*, pp. 27–51, 1996.

29. Eilenberg, S. and M. Schützenberger, "Rational sets in commutative monoids", *Journal of Algebra*, Vol. 13, No. 2, pp. 173 – 191, 1969.

30. Corson, J. M., "Extended finite automata and word problems", *International Journal of Algebra and Computation*, Vol. 15, No. 03, pp. 455–466, 2005.

31. Dassow, J. and V. Mitrana, "Finite automata over free groups", *International Journal of Algebra and Computation*, Vol. 10, No. 06, pp. 725–737, 2000.

32. Mitrana, V. and R. Stiebe, "Extended finite automata over groups", *Discrete Applied Mathematics*, Vol. 108, No. 3, pp. 287–300, 2001.

33. Kambites, M., "Formal languages and groups as memory", *Communications in Algebra*, Vol. 37, No. 1, pp. 193–208, 2009.

34. Render, E. and M. Kambites, "Rational subsets of polycyclic monoids and valence automata", *Inf. Comput.*, Vol. 207, No. 11, pp. 1329–1339, 2009.

35. Render, E., *Rational Monoid and Semigroup Automata*, Ph.D. Thesis, University of Manchester, 2010.

36. Elston, G. Z. and G. Ostheimer, "On groups whose word problem is solved by a counter automaton", *Theoretical Computer Science*, Vol. 320, No. 2–3, pp. 175 – 185, 2004.

37. Kambites, M., "Word problems recognisable by deterministic blind monoid automata", *Theoretical Computer Science*, Vol. 362, No. 1, pp. 232–237, 2006.

38. Elder, M., M. and G. Ostheimer, "On groups and counter automata", *International Journal of Algebra and Computation*, Vol. 18, No. 08, pp. 1345–1364, 2008.

39. Corson, J. M. and L. L. Ross, "Automata with counters that recognize word problems of free products", *International Journal of Foundations of Computer Science*, Vol. 26, No. 01, pp. 79–98, 2015.

40. Bishop-Ross, R., J. M. Corson and J. L. Ross, "Context-sensitive languages and G-automata", *International Journal of Algebra and Computation*, Vol. 27, No. 02, pp. 237–249, 2017.

41. Zetzsche, G., *Monoids as Storage Mechanisms*, Phd thesis, Technische Universität Kaiserslautern, 2016.

42. Cleary, S., M. Elder and G. Ostheimer, "The word problem distinguishes counter languages", *arXiv preprint math/0606415*, 2006.

43. Kargapolov, M. I. and J. I. Merzljakov, *Fundamentals of the Theory of Groups*, Springer-Verlag, 1979.

44. Brown, N. P. and N. Ozawa, *C*-Algebras and Finite-Dimensional Approximations*, Vol. 88, American Mathematical Soc., 2008.

45. Grigorchuk, R. I., "On growth in group theory", *Proceedings of the International Congress of Mathematicians*, Vol. 1, pp. 325–338, 1990.

46. Nivat, M. and J. F. Perrot, "Une généralisation du monoïde bicyclique", *Comptes Rendus de l'Académie des Sciences de Paris*, Vol. 271, pp. 824–827, 1970.

47. Nivat, M., "Sur les automates a mémoire pile", *Proc. of International Computing Symposium*, pp. 221–225, 1970.

48. Zetzsche, G., "Silent transitions in automata with storage", *International Colloquium on Automata, Languages, and Programming*, pp. 434–445, Springer Berlin Heidelberg, 2013.

49. La Harpe, P. D., *Topics in Geometric Group Theory*, The University Of Chicago Press, Chicago, 2000.

50. Glaister, I. and J. Shallit, "Automaticity III: polynomial automaticity and context-free languages", *Computational Complexity*, Vol. 7, No. 4, pp. 371–387, 1998.

51. Żak, S., "A Turing machine time hierarchy", *Theoretical Computer Science*, Vol. 26, No. 3, pp. 327 – 333, 1983.

52. Dehn, M., "Über unendliche diskontinuierliche Gruppen", *Mathematische Annalen*, Vol. 71, No. 1, pp. 116–144, 1911.

53. Mikhailova, K. A., "The occurrence problem for direct products of groups", *Dokl. Akad. Nauk SSSR*, Vol. 119, No. 6, pp. 1103–1105, 1958.

54. Paz, A., "Events which are not representable by a probabilistic automaton", *SIGACT News*, , No. 4, pp. 8–11, 1970.

55. Colcombet, T., J. Ouaknine, P. Semukhin and J. Worrell, "On reachability problems for low dimensional matrix semigroups", *arXiv preprint arXiv:1902.09597*, 2019.

56. Choffrut, C. and J. Karhumäki, "Some decision problems on integer matrices", *RAIRO-Theoretical Informatics and Applications*, Vol. 39, No. 1, pp. 125–131, 2005.

57. Potapov, I. and P. Semukhin, "Membership problem in GL(2, $\mathbb{Z}$) extended by singular matrices", *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 83, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

58. Potapov, I. and P. Semukhin, "Decidability of the membership problem for 2×2 integer matrices", *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 170–186, SIAM, 2017.

59. Ko, S.-K., R. Niskanen and I. Potapov, "On the identity problem for the special linear group and the Heisenberg group", *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, Vol. 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 132:1–132:15, 2018.

60. Bell, P. C. and I. Potapov, "On the undecidability of the identity correspondence problem and its applications for word and matrix semigroups", *International Journal of Foundations of Computer Science*, Vol. 21, No. 06, pp. 963–978, 2010.

61. Kambites, M., P. V. Silva and B. Steinberg, "On the rational subset problem for groups", *Journal of Algebra*, Vol. 309, No. 2, pp. 622–639, 2007.

62. Lohrey, M., "The rational subset membership problem for groups: a survey", *Groups St Andrews*, Vol. 422, pp. 368–389, 2013.

63. Zetzsche, G., "The emptiness problem for valence automata or: Another decidable extension of Petri nets", *International Workshop on Reachability Problems*, pp. 166–178, Springer, 2015.

64. Baumslag, G. and J. E. Roseblade, "Subgroups of direct products of free groups", *Journal of the London Mathematical Society*, Vol. 2, No. 1, pp. 44–52, 1984.

65. Stern, M. A., "Über eine zahlentheoretische Funktion", *Journal für die reine und angewandte Mathematik*, Vol. 55, pp. 193–220, 1858.

66. Brocot, A., "Calcul des rouages par approximation, nouvelle méthode", *Revue Chronométrique*, Vol. 3, pp. 186–194, 1861.

67. Graham, R., D. Knuth and O. Patashnik, *Concrete Mathematics: A foundation for computer science*, Addison-Wesley, 1989.

68. Garrity, T., "A multidimensional continued fraction generalization of Stern's diatomic sequence", *Journal of Integer Sequences*, Vol. 16, No. 2, p. 3, 2013.

69. Render, E. and M. Kambites, "Semigroup automata with rational initial and terminal sets", *Theoretical Computer Science*, Vol. 411, No. 7-9, pp. 1004–1012, 2010.

70. Goralčík, P. and V. Koubek, "On discerning words by automata", *International Colloquium on Automata, Languages, and Programming, ICALP'86*, Vol. 226 of

LNCS, pp. 116–122, Springer, 1986.

71. Yang, L., Z. Dang and O. H. Ibarra, "On stateless automata and P Systems", *Int. J. Found. Comput. Sci.*, Vol. 19, No. 5, pp. 1259–1276, 2008.

72. Ibarra, O. H., J. Karhumäki and A. Okhotin, "On stateless multihead automata: Hierarchies and the emptiness problem", *Theoretical Computer Science*, Vol. 411, No. 3, pp. 581–593, 2010.

73. Kutrib, M., H. Messerschmidt and F. Otto, "On stateless deterministic restarting automata", *35th Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM'09*, Vol. 5404 of *LNCS*, pp. 353–364, Springer, 2009.

74. Păun, G., "Computing with membranes", *Journal of Computer and System Sciences*, Vol. 61, No. 1, pp. 108–143, 2000.

75. Lehmer, D. H., "A note on trigonometric algebraic numbers", *The American Mathematical Monthly*, Vol. 40, No. 3, pp. 165–166, 1933.

76. Fernau, H. and R. Stiebe, "Regulation by valences", *International Symposium on Mathematical Foundations of Computer Science*, pp. 239–248, Springer, 1997.

77. Hoogeboom, H. J., "Context-free valence grammars-revisited", *International Conference on Developments in Language Theory*, pp. 293–303, Springer, 2001.

78. Zetzsche, G., "On the capabilities of grammars, automata, and transducers controlled by monoids", *Proceedings of the 38th International Conference on Automata, Languages and Programming - Volume Part II*, ICALP'11, pp. 222–233, Springer-Verlag, 2011.

79. Howie, J. M., *Fundamentals of Semigroup Theory*, Clarendon Oxford, 1995.